Raghunath Nambiar
Meikel Poess (Eds.)

Performance Evaluation, Measurement and Characterization of Complex Systems

LNCS 6417

Second TPC Technology Conference, TPCTC 2010
Singapore, September 2010
Revised Selected Papers

Springer

Lecture Notes in Computer Science 6417

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Raghunath Nambiar Meikel Poess (Eds.)

Performance Evaluation, Measurement and Characterization of Complex Systems

Second TPC Technology Conference, TPCTC 2010
Singapore, September 13-17, 2010
Revised Selected Papers

Springer

Volume Editors

Raghunath Nambiar
Cisco Systems, Inc., Server Access and Virtualization Business Unit
3800 Zankar Road, San Jose, CA 95134, USA
E-mail: rnambiar@cisco.com

Meikel Poess
Oracle Corporation, Server Technologies
500 Oracle Parkway, Redwood Shores, CA 94065, USA
E-mail: meikel.poess@oracle.com

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-18205-1 e-ISBN 978-3-642-18206-8
DOI 10.1007/978-3-642-18206-8
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2010942795

CR Subject Classification (1998): C.4, C.2, H.4, D.2, H.3, C.2.4

LNCS Sublibrary: SL 2 – Programming and Software Engineering

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

Over the past two decades, the Transaction Processing Performance Council (TPC) has had a significant impact on the computing industry's use of industry-standard benchmarks. Vendors use TPC benchmarks to illustrate performance competitiveness for their existing products, and to improve and monitor the performance of their products under development. Many buyers use TPC benchmark results as points of comparison when purchasing new computing systems.

The information technology landscape is evolving at a rapid pace, challenging industry experts and researchers to develop innovative techniques for evaluation, measurement and characterization of complex systems. The TPC remains committed to developing new benchmark standards to keep pace, and one vehicle for achieving this objective is the sponsorship of the Technology Conference on Performance Evaluation and Benchmarking (TPCTC). With this conference, the TPC encourages researchers and industry experts to present and debate novel ideas and methodologies in performance evaluation and benchmarking.

The first TPC Technology Conference on Performance Evaluation and Benchmarking (TPCTC 2009) was held in conjunction with the 35th International Conference on Very Large Data Bases (VLDB 2009) in Lyon, France during August 24–28, 2009.

This book contains the proceedings of the second TPC Technology Conference on Performance Evaluation and Benchmarking (TPCTC 2010), held in conjunction with the 36th International Conference on Very Large Data Bases (VLDB 2010) in Singapore during September 13–17, 2010 including 14 selected papers and two keynote papers.

The hard work and close cooperation of a number of people have contributed to the success of this conference. We would like to thank the members of TPC and the organizers of VLDB 2010 for their sponsorship; the members of the Program Committee and Publicity Committee for their support; and the authors and the participants who are the primary reason for the success of this conference.

November 2010

Raghunath Nambiar
Meikel Poess

TPCTC 2010 Organization

General Chair

Raghunath Nambiar (Cisco)

Program Committee Chair

Meikel Poess (Oracle)

Publicity Committee Chair

Nicholas Wakou (Dell)

Program Committee

Alain Crolotte (Teradata)
Badriddine Khessib (Microsoft)
Berni Schiefer (IBM)
Guogen Zhang (IBM)
Harumi Kuno (HP Labs)
Kai Sachs (TU Darmstadt)
Marco Vieira (University of Coimbra)
Masaru Kitsuregawa (University of Tokyo)
Mike Molloy (Dell)
Paul Larson (Microsoft Research)
Peter Thawley (Sybase)
Yicheng Tu (University of South Florida)

Publicity Committee

Jerrold Buggert (Unisys)
Matthew Lanken (Oracle)
Peter Thawley (Sybase)
Forrest Carman (Owen Media)
Michael Majdalany (LoBue & Majdalany Management Group)

Keynote

C. Mohan (IBM Almaden Research Center)

Invited Talk

Karl Huppler (IBM)

About the TPC

Introduction to the TPC

The Transaction Processing Performance Council (TPC) is a non-profit organization that defines transaction processing and database benchmarks and distributes vendor-neutral performance data to the industry. Additional information is available at `http://www.tpc.org/`.

TPC Memberships

Full Members

Full members of the TPC participate in all aspects of the TPC's work, including development of benchmark standards and setting strategic direction. The full member application can be found at
`http://www.tpc.org/information/about/app-member.asp`.

Associate Members

Certain organizations may join the TPC as associate members. Associate members may attend TPC meetings, but are not eligible to vote or hold office. Associate membership is available to non-profit organizations, educational institutions, market researchers, publishers, consultants, governments and businesses that do not create, market or sell computer products or services. The associate member application can be found at
`http://www.tpc.org/information/about/app-assoc.asp`.

Academic and Government Institutions

Academic and government institutions are invited join the TPC and a special invitation can be found at
`http://www.tpc.org/information/specialinvitation.asp`.

Contact the TPC

TPC
Presidio of San Francisco
Building 572B (surface)
P.O. Box 29920 (mail)
San Francisco, CA 94129-0920
Voice: 415-561-6272
Fax: 415-561-6120
Email: info@tpc.org

How to Order TPC Materials

All of our materials are now posted free of charge on our website. If you have any questions, please feel free to contact our office directly or by email at info@tpc.org.

Benchmark Status Report

The TPC Benchmark Status Report is a digest of the activities of the TPC and its technical subcommittees. Sign-up information can be found at the following URL: http://www.tpc.org/information/about/email.asp.

TPC 2010 Organization

Full Members

AMD
Bull
Cisco
Dell
Fujitsu
Fusion IO
HP
Hitachi
IBM
Ingres
Intel
Microsoft
NEC
Netezza
Oracle
ParAccel
Sybase
Syncsort
Teradata
Unisys
VMware

Associate Members

Ideas International
ITOM International Co
TTA

TPC 2010 Organization

Steering Committee

Karl Huppler (IBM), Chair
Charles Levine (Microsoft)
Jerrold Buggert (Unisys)
Raghunath Nambiar (Cisco)
Wayne Smith (Intel)

Public Relations Committee

Nicholas Wakou (Dell), Chair
Jerrold Buggert (Unisys)
Matthew Lanken (Oracle)
Peter Thawley (Sybase)
Raghunath Nambiar (Cisco)

Technical Advisory Board

Jamie Reding (Microsoft), Chair
Dave Steinhoff (ParAccel)
Matthew Emmerton (IBM)
Mike Nikolaiev (HP)
Nicholas Wakou (Dell)
Rick Freeman (Unisys)
Wayne Smith (Intel)

Table of Contents

Transaction Processing Performance Council (TPC): State of the Council 2010

Raghunath Nambiar[1], Nicholas Wakou[2],
Forrest Carman[3], and Michael Majdalany[4]

[1] Cisco Systems, Inc., 3800 Zanker Road, San Jose, CA 95134, USA
rnambiar@cisco.com
[2] Dell Inc., One Dell Way, Round Rock, TX 78682, USA
Nicholas_Wakou@dell.com
[3] Owen Media, 3130 E. Madison St., Suite 206, Seattle, WA 98112, USA
forrestc@owenmedia.com
[4] LoBue & Majdalany Magt Group, 572B Ruger St. San Francisco, CA 94129, USA
majdalany@lm-mgmt.com

Abstract. The Transaction Processing Performance Council (TPC) is a non-profit corporation founded to define transaction processing and database benchmarks and to disseminate objective, verifiable performance data to the industry. Established in August 1988, the TPC has been integral in shaping the landscape of modern transaction processing and database benchmarks over the past twenty-two years. This paper provides an overview of the TPC's existing benchmark standards and specifications, introduces two new TPC benchmarks under development, and examines the TPC's active involvement in the early creation of additional future benchmarks.

Keywords: Transaction Processing Performance Council, Industry Standard Benchmarks, Transaction Performance, Decision Support System Performance, Pricing Specification, Energy Specification.

1 Introduction

Originally formed in 1988, the Transaction Processing Performance Council (TPC) [1] is a non-profit corporation tasked with defining transaction processing and database benchmarks, and disseminating objective, verifiable performance data to the computing industry. The TPC was originally founded in response to a growing trend in "benchmarketing," or an attempt by vendors to publish questionable benchmark results in order to increase hardware sales. Without independent and objective oversight, a number of vendors created highly tailored workload environments, while deleting crucial benchmark requirements in order to improve specific performance results. This, in turn, enabled these vendors to exaggerate performance marketing claims in order to boost hardware sales. The need for a vendor-neutral organization tasked with disseminating objective, verifiable performance data quickly became apparent, ultimately culminating in the formation of the TPC. Both vendors and end-users have come to rely on TPC benchmarks to provide real-world data that is backed by a stringent and independent review process. Vendors use TPC benchmarks to

R. Nambiar and M. Poess (Eds.): TPCTC 2010, LNCS 6417, pp. 1–9, 2011.

illustrate performance competitiveness for their existing products, and to improve and monitor the performance of their products under development [2]. End-users oftentimes use TPC benchmark results as points-of-comparison when purchasing new computing systems. Further end-user and vendor benefits are outlined below:

1) *Objective price, price/performance and energy/performance comparisons.* TPC testing requires vendors to specify their hardware and software components, and to disclose associated costs and maintenance fees for three years. Additionally, the new TPC-Energy specification will encourage the comparison of watts/performance between disparate hardware architectures.
2) *Direct comparison of different vendor architectures on an apples-to-apples basis.* In the past, purchasers spent substantial time and resources defining customized benchmarks. This was the only means of directly comparing performance between different products. The fairness and objectivity of such benchmarks were, however, frequently questioned by the vendor community. Today, end-users and vendors have accepted standardized TPC benchmarks, allowing a streamlined, objective means of comparing disparate computing architectures.
3) *An independent auditing process.* TPC-certified auditors verify all results before vendors can publish benchmark results. The TPC also encourages a peer review process after each result is published, during which any TPC member can challenge results for up to sixty days.
4) *TPC benchmarks encourage both hardware and software improvements.* TPC benchmarks are well-understood, and enable engineers to eliminate hardware and software bottlenecks. This results in real-world performance improvements for end-users.
5) *A credible means of evaluating complete systems, subsystems and/or processors.* TPC benchmarks model realistic workloads across the end-to-end business computing spectrum. Many non-TPC benchmarks only measure the hardware performance of a given processor and memory subsystem.

TPC benchmarks have raised the bar for what the computing industry has come to expect in terms of benchmarks themselves. They enable the direct comparison of performance, price, price/performance, and energy/performance between servers manufactured by different vendors.

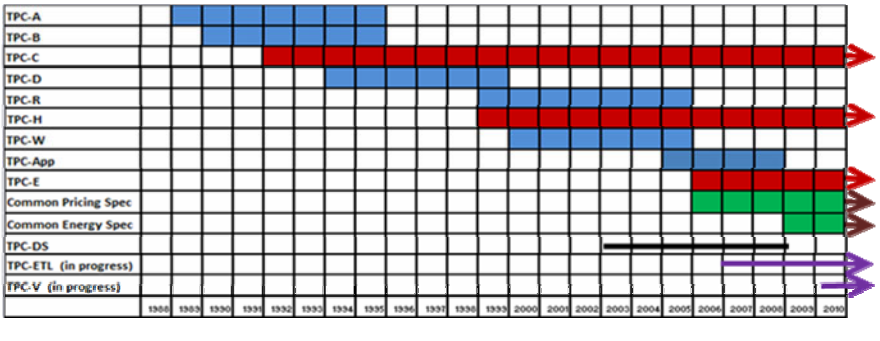

Fig. 1. TPC Benchmark Life spans

To date, the TPC has approved a total of nine independent benchmarks [2] [3], as shown in Figure 1. Of these benchmarks, TPC-E, TPC-C and TPC-H are the current active ones and are widely being used by the industry.

Benchmark specifications currently under development are TPC-Virtualization and TPC-ETL (extract/transform/load). TPC also developed a benchmark specification TPC-DS [4], the next generation Decision Support benchmark, but is unable to reach consensus. Additionally, the TPC has introduced two specifications, including a common pricing specification (TPC-Pricing) across all its benchmarks in 2006, and a common energy specification (TPC-Energy), across all its benchmarks in 2009. Each of these specifications is outlined below:

The following section provides an overview of the TPC's current industry standard benchmarks. Section 3 examines the TPC-Pricing specification and TPC-Energy specification in detail. Section 4 provides insight into benchmark standards under developments. Finally, Section 5 reviews the TPC's Technology conference initiative and examines the future of TPC benchmarks.

2 Current Industry Standards

Current industry standard benchmarks are TPC-E, TPC-C and TPC-H each addressing distinct industry requirements. TPC-E is the new transaction processing benchmark gaining momentum, while TPC and TPC-H benchmarks continue to be a popular yardstick for comparing transaction processing performance and decision support performance respectively. The longevity of these benchmarks means that hundreds of results are publicly available over a wide variety of hardware and software platforms.

2.1 TPC-E Benchmark

First released in March 2007, the TPC-E benchmark [5] simulates the OLTP workload of a brokerage firm with customers who generate transactions related to trades, account inquiries and market research. The brokerage firm, in turn, interacts with financial markets to execute orders on behalf of the customers and updates relevant account information. TPC-E is scalable, meaning that the number of customers defined for the brokerage firm can be varied to represent the workloads of different-size businesses. The benchmark also defines the required mix of transactions the benchmark must maintain, and is given in transactions per second (tpsE). It specifically refers to the number of Trade-Result transactions the server can sustain over a period of time. Although the underlying business model of TPC-E is a brokerage firm, the database schema, data population, transactions, and implementation rules have been designed to be broadly representative of modern OLTP systems. One of the primary goals of the TPC-E benchmark is to enhance the complexity of a schema in a way that captures the best practices of a modern day OLTP system. The mixture and variety of transactions being executed on the benchmark system are designed to capture the characteristic components of a complex system. Different transaction types are defined to simulate the interactions of a brokerage firm, with its customers as well as its business partners.

The adoption rates of TPC-E are beginning to increase, since the benchmark successfully models a given workload , and its features are easy to use. One of the main concerns about TPC-E has been the CPU/IO balance. It was expected that the TPC-E workload would use significantly fewer disks than the widely used TPC-C benchmark, thereby cutting down benchmarking costs. A look at the most recent results, however, shows that sponsors are configuring more storage than is required to run the benchmark. This is an indication that the system has to generate a lot more IO, at increasing speeds, in order to saturate the CPUs. Since CPU processing power follows Moore's law, the number of disks required is likely to get even higher unless sponsors consider using emerging but costly technologies like Solid State Disks (SSD).

2.2 TPC-C Benchmark

Like TPC-E, the TPC-C benchmark [6] is an on-line transaction processing (OLTP) benchmark, although these two benchmarks have very different workloads. When it was introduced, TPC-C was more complex than previous OLTP benchmarks because of its multiple transaction types, more complex database and overall execution structure. TPC-C involves a mix of five concurrent transactions of different types and complexity either executed on-line or queued for deferred execution. The database is comprised of nine types of tables with a wide range of record and population sizes. TPC-C is measured in transactions per minute (tpmC). TPC-C simulates a complete computing environment where a population of users executes transactions against a database. The benchmark is centered around the principal activities (transactions) of an order-entry environment such as a warehouse. These transactions include entering and delivering orders, recording payments, checking the status of orders, and monitoring the level of stock at the warehouses. While the benchmark portrays the activity of a wholesale supplier, TPC-C is not limited to the activity of any particular business segment, but rather represents any industry that must manage, sell, or distribute a product or service.

TPC-C is arguably the most popular TPC benchmark and is considered as the yardstick for OLTP performance. There have been approximately 750 results submitted since the first one in 1994. A look at the first result, and the current top performance and price/performance results, shows a drop in price/performance by a factor of 1300 and an improvement in performance by a factor of 15,900. Based on the trends in the computer industry with server systems becoming more powerful and less costly, these figures show that TPC-C has managed to successfully capture these trends within the industry. In order to maintain TPC-C's applicability to systems of different capacity, TPC-C implementations must scale both the number of users and the size of the database proportionally to the computing power of the system to be measured. The scalability of its workload and the simplicity of its performance metric (transactions per minute) have lead to more TPC-C published results than any other TPC benchmark. The cost of TPC-C benchmark configurations have grown alarmingly high, mainly as a result of the large disk farms that have to be deployed to saturate a modern day server systems. With hard disk drive performance lagging behind that of CPUs (whose processing power follows Moore's Law), the cost and complexity of TPC-C configurations is likely to grow even more. The use of SSDs in data centers and benchmarking efforts is likely to benefit TPC-C, as it will cut down on the size and

complexity of storage systems. However, it will be some time before SSDs are economical enough to be commoditized.

2.3 TPC-H Benchmark

Originally released in May 1999, the TPC-H benchmark [7] is a decision support benchmark. It consists of a suite of business oriented ad-hoc queries and concurrent data modifications. The queries and the data populating the database have been chosen to have broad industry-wide relevance. This benchmark illustrates decision support systems that examine large volumes of data, execute queries with a high degree of complexity, and give answers to critical business questions.

The TPC-H benchmark represents decision support environments where users don't know which queries will be executed against a database system; hence, the "ad-hoc" label. Pre-knowledge of the queries may not be used to optimize the DBMS system. Consequently, query execution times can be very long.

The performance metric reported by TPC-H is called the TPC-H Composite Query-per-Hour performance metric (QphH@Size), and reflects multiple aspects of the capability of the system to process queries. These aspects include the selected database size against which the queries are executed, the query processing power when queries are submitted by a single stream, and the query throughput when queries are submitted by multiple concurrent users. The TPC-H Price/Performance metric is expressed as $/QphH@Size.

TPC-H uses scale factors to distinguish the workload that best captures the capabilities of the benchmark System Under Test (SUT). This is a realization that larger systems run a more demanding workload than smaller systems. Scale Factors indicate the raw data size to be loaded in Gigabytes. They are chosen from the following set of 10 fixed scale factors { 1, 10, 30, 100, 300, 1,000, 3,000, 10,000, 30,000, 100,000 }. For instance, a scale factor of 300 means that the raw data size of the database to be loaded is 300GB.

TPC-H has been widely accepted in the industry as the benchmark-of-choice for decision support systems. Its use grew steadily from 1999, when the first result was published, to 2005 when the most number of results were published. In recent years, TPC-H has attracted many new members to the TPC, and vendors have published TPC-H results across a variety of platforms.

3 Specifications Consistent across All Standards

TPC benchmarks are intended to provide a fair comparison of performance, price-performance and energy efficiency of various vendor implementations. The TPC-Pricing specification and TPC-Energy specifications are consistent across all current standards and are expected to be consistent across future specifications as well. The TPC-Pricing specification is intended to provide consistent methodologies for reporting the purchase price of the benchmarked system, the licensing of software used in the benchmark, contracts for maintenance and general availability. The TPC-Energy specification augments the existing TPC benchmarks by adding the methodology and requirements for including and reporting energy consumption.

The primary metrics defined by these standards are three year cost of ownership, price/performance, availability date and Watts/performance.

3.1 TPC-Pricing Specification

With the establishment of the TPC's first benchmark, TPC-A, in 1989, the TPC set the standard for the inclusion of price and price/performance in benchmarks. Each benchmark had its own pricing requirements. In 2004, the TPC-Pricing subcommittee was formed to recommend revisions to the existing pricing methodology. The TPC-Pricing subcommittee elected to develop a single pricing specification, consistent across all TPC benchmarks, which was approved in 2008.

The TPC-Pricing specification [8] is designed to guide customers, vendors implementing TPC benchmarks, and TPC auditors on what is acceptable pricing for the purposes of publication. The pricing methodology reflects the price for the purchase of the benchmark SUT, the licensing of software used in the benchmark and the contracts for maintenance. The TPC-Pricing specification also establishes an availability metric, which provides information on whether a specific benchmark configuration can be purchased immediately, or whether some of the components of the configuration might not be available for some time. The availability requirements also limit the length of time before a promised result must be fully available. There may be some restrictions on pricing for publication (such as excluding sales and closeouts) that are different from some business transactions that actually take place in the marketplace, but these restrictions are intended to make publication both tractable and comparable during the lifetime of the publication for the majority of customers and vendors.

The TPC-Pricing Subcommittee continues to review and to recommend future revisions to the existing pricing methodology, so that prices used in published TPC results remain relevant and consistent for all current and future TPC benchmarks.

3.2 TPC-Energy Specification

In the past, performance and price/performance were the key criteria in data center purchasing decisions. In recent years, energy efficiency has become another important factor in evaluating systems and, as a result, the TPC-Energy committee was formed in 2007. The TPC-Energy specification [9], which was approved in 2009, augments the existing TPC benchmarks with energy metrics and methodology, and requirements for including and reporting them.

The primary metric as defined by TPC-Energy is in the form of "Watts per performance" where the performance units are particular to each TPC benchmark (for example, Watts per ktpmC for a TPC-C result). The measuring and publication of TPC-Energy metrics are optional and are not required to publish a TPC benchmark result. With the TPC-Energy metric, customers can identify systems that meet their price, performance and energy requirements via the TPC Web site.

To facilitate the implementation of the TPC-Energy specification, and to help TPC benchmark sponsors reduce costs, the TPC provides a software suite, the Energy Measurement System (EMS). The EMS provides services such as power instrumentation interfacing, power and temperature logging and report generation. Even though reporting energy metrics is optional, competitive demands are expected to encourage vendors to include them.

4 Industry Standards under Development

The TPC has two new benchmark development committees chartered to develop new benchmark standards: TPC-ETL and TPC-Virtualization.

4.1 TPC-ETL Committee

The TPC-ETL benchmark committee was formed in 2008 to develop a standard [10] [11] for comparing the performance of ETL systems. The objectives of this benchmark are to provide a workload that is scalable over a wide range of dataset sizes, methodologies and metrics, and to compare the performance, price/performance and energy efficiency of ETL systems based on industry-relevant scenarios. The TPC-ETL benchmark is expected to address the strong industry demand for a well designed ETL benchmark, and to benefit customers evaluating ETL systems, ETL tools, vendors and system vendors. Once there is a common basis for comparing ETL systems and ETL tools, vendors will improve their products to compete on the basis of the benchmark.

The ETL benchmark committee completed draft transformation specifications, and has started work on data generation. Additionally, some member companies are now working on benchmark transformation test implementations.

4.2 TPC-Virtualization Committee

Virtualization was identified as a pressing area for benchmark development during the TPC's first Technology Conference on Performance Evaluation and Benchmarking in 2009. As a result, the TPC-Virtualization Work Group [12] [13] was formed to respond to the growth in virtualization technology and capture some of the common-case usage of virtualization with database workloads, in a well-designed benchmark for virtual environments.

The TPC-Virtualization Work Group is currently defining a benchmark with the following design goals in mind:

- Incorporates a database-centric workload
- Features a resource-intensive workload that stresses the virtualization layer
- Emphasizes storage and networking I/O in a virtualized environment
- Has a timely development cycle

The TPC-Virtualization Work Group has made substantial progress in defining the framework of a new benchmark, and both major systems and software vendors are represented in this group.

5 TPC Technology Conference and the Future of Benchmark Development

The technology landscape is continually evolving and challenging industry experts and researchers to develop innovative techniques to evaluate and benchmark computing systems. The TPC remains committed to developing highly relevant benchmark

standards and will continue to develop new benchmarks to keep pace. One vehicle for achieving these objectives is the sponsorship and ongoing commitment towards hosting the Technology Conference on Performance Evaluation and Benchmarking (TPCTC) [14]. The TPCTC provides industry experts and researchers with a forum to present and debate novel ideas and methodologies in performance evaluation, measurement and characterization.

The first conference in this series [14] was conducted in conjunction with the 35th International Conference on Very Large Databases (VLDB) on August 24-28, 2009 in Lyon, France. The proceedings of this conference are available from Springer-Verlag at http://www.springer.com/computer/hardware/book/978-3-642-10423-7 [14].

Topics of future interest include, but are not limited to: appliance, business intelligence, cloud computing, complex event processing, data compression, database optimizations, disaster tolerance and recovery, energy and space efficiency, green computing, hardware innovations, high speed data generation, hybrid workloads, software management and maintenance, unstructured data management, virtualization and very large memory systems. Topics also include enhancements to existing TPC benchmark standards.

Identifying, defining and ultimately publishing new benchmark standards, demonstrates the TPC's commitment towards meeting the needs of an ever-changing industry. To this end, the TPC's Technology Conference on Performance Evaluation and Benchmarking is a concrete step towards identifying and defining the benchmarks of the future. The TPC actively encourages the submission of benchmark ideas, and wider participation from companies, government organizations and the research community. Institutions interested in developing new benchmark standards and enhancing existing benchmark standards are invited to join the TPC [15].

Acknowledgements

The authors would like to thank the past and present members of the TPC for their contribution to specifications and documents referenced in this paper.

References

1. Transaction Performance Council website (TPC), http://www.tpc.org
2. Nambiar, R.O., Lanken, M., Wakou, N., Carman, F., Majdalany, M.: Transaction Processing Performance Council (TPC): Twenty Years Later – A Look Back, a Look Ahead. In: Nambiar, R.O., Poess, M. (eds.) TPCTC 2009. LNCS, vol. 5895, pp. 1–10. Springer, Heidelberg (2009)
3. Dunn, A.: Twenty Years of TPC Benchmarks, http://www.ideasint.blogs.com/ideasinsights/2008/10/twenty-years-of.html
4. Poess, M., Nambiar, R.O., Walrath, D.: Why You Should Run TPC-DS: A Workload Analysis. In: VLDB 2007, pp. 1138–1149 (2007)
5. TPC-E Specification, http://www.tpc.org/tpce/default.asp
6. TPC-C Specification, http://www.tpc.org/tpcc/default.asp
7. TPC-H Specification, http://www.tpc.org/tpch/default.asp

8. TPC-Pricing Specification, http://www.tpc.org/pricing/
9. TPC-Energy Specification, http://www.tpc.org/tpc_energy/
10. TPC-ETL Draft Specification, http://www.tpc.org/reports/status/default.asp
11. Wyatt, L., Caufield, B., Pol, D.: Principles for an ETL Benchmark. In: Nambiar, R.O., Poess, M. (eds.) TPCTC 2009. LNCS, vol. 5895, pp. 183–198. Springer, Heidelberg (2009)
12. TPC-V Draft Specification, http://www.tpc.org/reports/status/default.asp
13. Bose, S., Mishra, P., Sethuraman, P., Taheri, R.: Benchmarking Database Performance in a Virtual Environment. In: Nambiar, R.O., Poess, M. (eds.) TPCTC 2009. LNCS, vol. 5895, pp. 167–182. Springer, Heidelberg (2009)
14. Nambiar, R.O., Poess, M.: Performance Evaluation and Benchmarking. Springer, Heidelberg (2009)
15. Invitation to Join the TPC, http://www.tpc.org/information/about/join.asp

Liquid Benchmarks: Towards an Online Platform for Collaborative Assessment of Computer Science Research Results

Sherif Sakr[1] and Fabio Casati[2]

[1] NICTA and University of New South Wales, Sydney, Australia
ssakr@cse.unsw.edu.au
[2] University of Trento, Trento, Italy
casati@disi.unitn.it

Abstract. Experimental evaluation and comparison of techniques, algorithms, approaches or complete systems is a crucial requirement to assess the practical impact of research results. The quality of published experimental results is usually limited due to several reasons such as: limited time, unavailability of standard benchmarks or shortage of computing resources. Moreover, achieving an independent, consistent, complete and insightful assessment for different alternatives in the same domain is a time and resource consuming task in addition to its requirement to be periodically repeated to maintain its freshness and being up-to-date. In this paper, we coin the notion of *Liquid Benchmarks* as online and public services that provide collaborative platforms to unify efforts of peer researchers from all over the world to simplify their task in performing high quality experimental evaluations and guarantee a transparent scientific crediting process.

1 Introduction

The last two decades have seen significant growth in the number of scientific research publications. An important characteristic of Computer Science research is that it produces artifacts other than publications, in particular software implementations. The continuous existence of performance improvement claims from researchers have called for the necessity of applying experimental evaluation and comparison techniques between *competing alternative* implementations of algorithms, approaches or complete systems in order to assess the practical impact and benefit of the research results. While most research publications present experimental results to evaluate/compare their proposed scientific contributions, the quality of such experimental results are usually limited. That can happen due to several reasons such as: insufficient effort or time, unavailability of suitable test cases or any other resource constraints. Moreover, researchers are usually focusing on reporting the experimental results of the *good sides* of their work which may not reflect the whole picture of the real-world scenarios. Moreover, it becomes quite difficult to understand and assess the performance implications of the design decisions for a given approach.

R. Nambiar and M. Poess (Eds.): TPCTC 2010, LNCS 6417, pp. 10–24, 2011.

In practice, performing an *independent* and *large scale* benchmarking study in a certain domain is usually not an easy task as it requires a lot of effort and time due to several reasons such as: unavailability of standard benchmarks, unavailability of public access to the implementations for some techniques which are described in the research literature in addition to the constraints of having the different configuration of computing resources/environments that reflect the wide spectrum of different real-world scenarios. The recent advances in Web technologies have created new work environments that provide great opportunities to tackle the above mentioned problems. In particular, we envision our solution by combining the facilities provided by three main technologies:

1. *Cloud Computing* [7] as an efficient way for broad sharing of computer software and hardware resources via the Internet in an elastic way. Therefore, we can virtually have unbounded storage space and computing power.
2. *Software As A Service* (SAAS) [14] as an effective software distribution model in which applications are hosted by a service provider and made available to end-users over the Internet. Therefore, it does not require *each* end-user to manually download, install, configure, run or use the software applications on their own computing environments.
3. *Collaborative and Social Web Applications* [11] as a new generation of applications that support human interaction and enabled with Web 2.0 capabilities (e.g. tagging, wikis, blogs, forums) that offer a great flexibility in the ability of building online communities between groups of people who share the same interests (peers) where they can interact and work together in an effective and productive way.

Surprisingly, the world of Computer Science research has not been able so far to exploit these available opportunities to form and organize driving forces to tackle the above mentioned problems and produce functional and widely-accepted collaborative experimental evaluation and assessment environments. Although the scientific community has become increasingly using wikis and personal/shared blogs (e.g. Database Column[1], DBMS2[2], SemWebTec[3]) to share and discuss their findings, there is a still long way to go for achieving effective and collaborative innovation processes.

The *LiquidPub* project[4] proposes a paradigm shift in the way scientific knowledge is created, disseminated, evaluated and maintained. This shift is enabled by the notion of Liquid Publications, which are evolutionary, collaborative, and composable scientific contributions. Many Liquid Publication concepts are based on a parallel between scientific knowledge artifacts and software artifacts. In this position paper, we coin one of the main notions of this project, *Liquid Benchmarks*, which represents a first step towards building online and public services

[1] http://databasecolumn.vertica.com/
[2] http://www.dbms2.com/
[3] http://semwebtec.wordpress.com/
[4] http://project.liquidpub.org/

that provide robust platforms for collaborative construction of independent, comprehensive, large scale and continuously evolving benchmarks for any research aspect in the Computer Science field. The main principles of our solution are:

1. Leveraging the recent advances of the Web technologies and the Social Web aspects in facilitating the process of building focused communities of scientists that are able to collaboratively develop effective benchmarks in an easy way.
2. Providing the infrastructure for scientific researchers to evaluate the *impact* of their new contributions - in addition to the contributions of others - and draw their conclusions in an effective way with minimum effort (no installation or configuration work for software solutions).
3. Establishing an *open* and *transparent* platform for scientific *crediting* process that is based on collaborative community work in various forms such as: comments, feedbacks, scientific analysis,..., etc.

The remainder of this paper is organized in the following manner. In Section 2, we motivate the importance of our proposal by sample scenarios. Section 3 gives an overview of some of the main challenges for building conclusive and trustable experimental evaluations in the Computer Science filed. Section 4 describes the conceptual model and the main entities of the Liquid Benchmarks. The architecture for implementing the Liquid Benchmarks is presented in Section 5 before we conclude the paper with a prospect on future work in Section 6.

2 Motivating Scenarios

To motivate the importance of our proposed *Liquid Benchmark* platform, we present the following two sample scenarios.

Scenario 1. *John* is a bioinformatics doctoral student. He and his supervisor are planning to develop new efficient mechanisms for querying biological graph databases. *John* has been advised by his supervisor to start his work by surveying the literature and performing an experimental assessment to compare the performance of the state-of-the-art approaches. In his way to achieve this task, *John* has been overwhelmed with a large number of scientific proposals in the literature to solve the problem. After an extensive search process, *John* was able to have access to the implementation of some proposals while he used his skills to re-implement some of the unavailable but interesting proposals in the literature. After one year of work, *John* was able to perform a study that compares between some approaches which helped him to get some insights for achieving his primary goal. It is apparent that this task is quite time and effort consuming (in addition to other general challenges that will be discussed in Section 3). The quality of the results of this task is subject to (and limited by) the amount of time, effort and attention given by *John* (and his supervisor) during the investigation process. In practice, most of Computer Science doctoral students (in different domains) go through a similar process at the initial stages of their doctoral program. Future

doctoral students (all over the world) will not be able to make use of, build on or improve *John*'s work unless there is an effective way to share/collaborate on such type of experimental assessment tasks.

Scenario 2. *Alice* is an active researcher in the Web data management area. At some stage, he was interested in comparing the state-of-the-art of the compression techniques for XML datasets. He spent 5 months in 1) building a large corpus of XML datasets with different characteristics. 2) getting access to some of the implemented XML compressor in addition to implement some of the techniques which we described in the literature 3) performing an extensive experiment to compare the performance of the different proposed solution over the XML corpus and analyze the results 4) publishing the whole material of his experiments in addition to the results into a public web in addition to writing a journal paper to disseminate the experiences of his study.

After its release, *Alice*'s study has attracted a lot of interest from researchers in the same domain where some of them has exchanged some message with *Alice* to discuss some items in the experimental results or to get some help in repeating some parts of the experiment. However, these exchanged messages remained offline in *Alice*'s inbox. After sometime, Alice has changed his affiliation and his research interest moved to some domain. Thus, he become less responsive to messages from researchers in the XML compression domain about his benchmark. Moreover, the outcome of his experimental study has become out-of-date after the appearance of new research contributions and the improvement of previously investigated approach. Utilizing *Alice*'s effort in a cumulative fashion call for collaborative efforts from different researchers in the XML compression domain in addition to a suitable platform to support their activities.

3 Benchmarking Challenges in Computer Science

In this section, we give an overview of some of the main *challenges* for building conclusive and trustable experimental evaluation studies in the field of Computer Science as follows.

– *Not enough standard benchmarks are available or widely-used:* A benchmark is a *standard* test or set of tests that is used to evaluate/compare alternative approaches that have a common aim to solve a specific problem. Unavailability of a standard benchmark in a certain domain makes the job of researchers hard to evaluate/comprare their work and leads to having several adhoc experimental results in the literature. A benchmark usually consists of a motivating scenario, task samples and a set of performance measures. In practice, very few benchmarks were able to achieve big success in their communities. For example, in the database domain the following benchmarks have attracted a lot of interest: 1) The TPC group of benchmarks for database performance for transaction processing [3]. 2) The oo7 benchmark [9] as a standard benchmark for object-oriented databases.

3) The XML Benchmark Project (XMark) [4] as a mean to assess the performance characteristics of XML Data Management Systems. On the other side, there are still many other areas which have a crucial need for constructing benchmarks that address the needs of researchers in evaluating their contributions (e.g. RDF databases, graph databases, scientific databases, NOSQL databases) [23]. In principle, for any benchmark to be successful, it must gain wide acceptance by its target community. Therefore, the motivating scenario should be *simple*, the set of testing tasks and performance metrics should be *generic* and *complete* [10]. Moreover, these standard benchmarks need to satisfy other general and important qualities such as *relevance*, *portability*, *extensibility* and *scalability* [13]. In practice, it is difficult that a single benchmark can represent all usage models and achieve all these quality goals. Therefore, some domains require designing *microbenchmakrs* [8] that have deep focus in a certain direction. In general, a well designed benchmark in a certain domain is very beneficial to the active researchers in that domain as it forms the common basis for evaluation and comparing their research contributions. Hence, they become able to identify the pros and cons of their approach which help in guiding their improvement plans. However, designing a successful benchmark is a quite challenging task which is usually not easily achievable by a single author or research group. In an ideal world, simplifying and improving the task of building standard successful benchmarks can be achieved through collaborative efforts between peer researchers with the same fields.

– *Limited repeatability of published results:* In an ideal world of Computer Science research, researchers describe the core of their contributions in the paper and then make the source codes/binaries of their implementation in addition to the experimental datasets available for other researchers to be reused for repeating the published results in their paper. Such ideal process can provide many benefits. For instance, other researchers can *independently* evaluate the performance of provided implementation with other data sets, validate the paper claims and ensure that there is no hidden conditions that may affect the performance. Moreover, they can use these available implementations as a great starting point to compare with and evaluate their own developed solutions. An interesting example for such independent evaluation studies is the work of Sidirourgos et al. [22] where they have reported about an independent assessment of the published result by Abadi et al. in [5] which described an approach for implementing a vertically partitioned DBMS for Semantic Web data management. The results of this independent assessment revealed many interesting issues. For example, in [5] Abadi et al. reported that the performance of binary tables is superior to that of the clustered property table for processing RDF queries while Sidirourgos et al. [22] reported that even in column-store database, the performance of binary tables is not always better than clustered property table and depends on the characteristics of the data set. Moreover, the experiments of [5] reported that storing RDF data in column-store database is better than that of row-store database while [22]

experiments have shown that the gain of performance in column-store database depends on the number of predicates in a data set.

Unfortunately, the research world does not always follow the ideal view. For example, in [19] we conducted an experimental evaluation study for the state-of-the-art of XML compression tools [1]. The results of our study have shown that many tools presented in the literature have no available implementations. Clearly, that prevented us from ensuring the repeatability of the reported numbers in the literature and hindered the possibility of performing complete comparison between all of the proposed approaches. Recently, some groups started to organize open challenges in different domains (e.g. Semantic Web Challenge[5], Semantic Web Service Challenge[6]). Moreover, recent editions of SIGMOD conference have offered the possibility for authors of published papers to test their programs against the experimental data to verify the published experimental results. However, the repeatability report of SIGMOD 2008 has shown limited success to achieve the goal due to several reasons [16].

– *Constraints of computing resources:* In some domains, conducting experimental evaluations may require huge computing resources. Moreover, conducting experimental evaluations may require using *different* settings for the computing environments in a manner that is similar to different types of real-world environments. Such computing resources requirements may be not available for researchers in their home environments/labs which can prevent or limit their ability to do insightful experimental studies. For example, Pavlo et al. [18] have presented an experimental comparison between MapReduce and parallel databases in terms of their performance on executing *large-scale* data analysis tasks and their development complexity. Repeating the experiments of this paper by other researchers is not an easy task (due to the configuration of the testing environment) as the original experiments have been run over a cluster of 100 nodes. In principle, achieving a *fair* and *apples-to-apples* comparison between any two alternative scientific contributions requires performing their experiments using *exactly* the same computing environments. Moreover, it is important that a benchmark exercise hardware components and subsystems in a meaningful and realistic way. In an ideal word, researchers should have access to shared computing environments where they can evaluate/compare their contributions consistently. The suitable configuration of these testing computing environments can be also decided *collaboratively*.

– *Continuous evolution of the state-of-the-art:* Conducting an *independent* experimental evaluation study for the-state-of-the-art in any domain is a very useful but usually not an easy task that requires considerable work, time, and resources. It can require designing different scenarios, selecting different datasets and evaluating different performance metrics. Some journals such

[5] http://challenge.semanticweb.org/
[6] http://sws-challenge.org/wiki/index.php/Main_Page

as the *Elsevier Performance Evaluation* Journal[7] focus their article around such type of experimental evaluation work. Recently, VLDB conference has introduced a new track for experimental analysis papers that focus on understanding the drawbacks and advantages of different alternative approaches in the same domain. Although such types of publications are important, they suffer from a main problem in that they represent *snapshots* for the state-of-the-art at the time of their preparation. However, by default the research contributions in any field are always *dynamic* and *evolving*. For example, new approaches that tackle the same research problem of a previously published snapshot paper can be introduced or the performance of previously assessed approaches can improve. Therefore, such papers can go out-of-date after a relatively short time of their publication. Assuming that the results of such experimental studies can be maintained on web pages, *continuous* maintenance of the published results may require too much effort from their authors who may loose interest in redoing the same job after sometime. Finally, it is not preferable in such very dynamic environment to spend several years in producing a set of benchmarking results for a certain research domain.

4 Conceptual Model

The main goal of our liquid benchmark proposal is to tackle the above mentioned challenges by building an online collaborative platform that can simplify the task of peer researchers in performing high quality experimental evaluations. The services of this platform will provide peer researchers with many mutual facilities such as:

- Building repositories of competing implementations where these implementations are running as software services with no installation or configuration requirements at the users side.
- Sharing testing computing environments.
- Collaboratively discussing, defining and evolving the specifications of standard benchmarks to evaluate the competing implementations.
- Allowing end-users to easily create and run testing experiments and share their results.

Figure 1 gives an overview of the conceptual model for the main entities of the Liquid Benchmarks. In this model, we differentiate between two types of users: *developer user* (benchmark developing committee) and *normal user*. Developer users represent the set of researchers who have the privilege to participate in the collaborative environment for defining the configurations of the different components of the benchmark (e.g. tasks, datasets, testing environments, evaluated implementation) while normal users are only allowed to use the defined configuration of the benchmark to run their test experiments. However, normal users can be optionally allowed to do some configuration tasks such as: uploading their

[7] http://www.elsevier.com/locate/peva

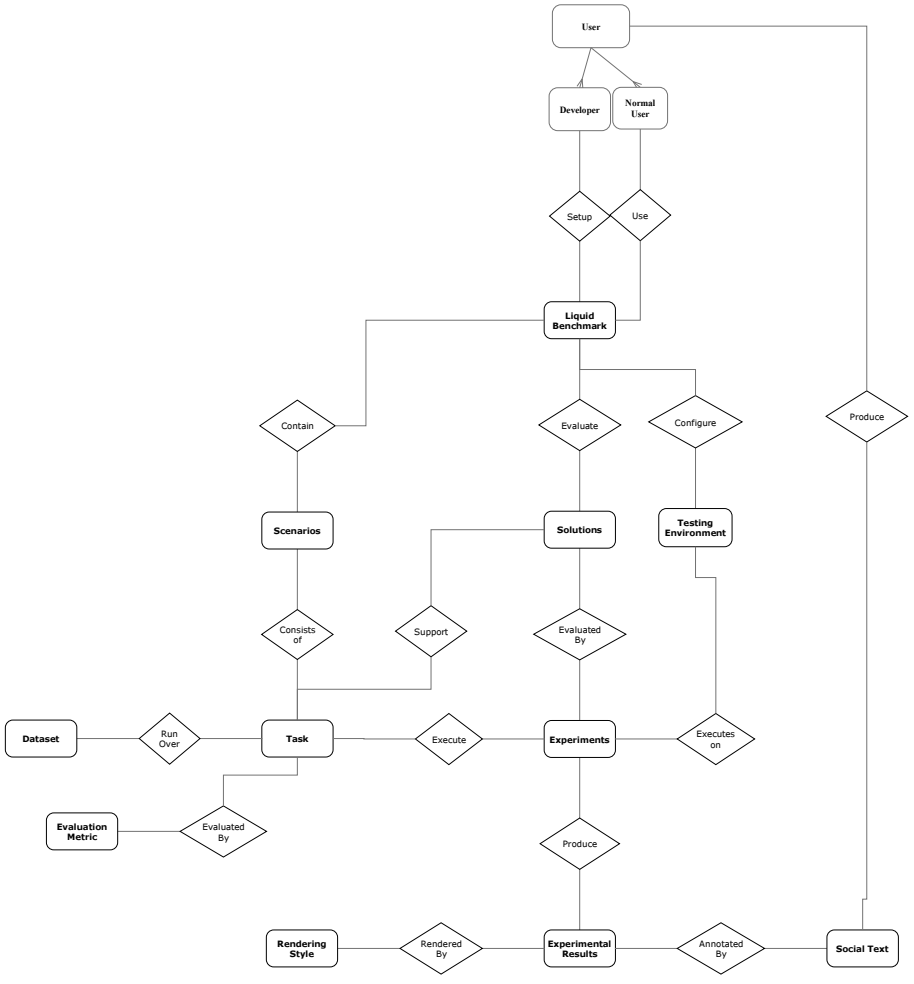

Fig. 1. Conceptual Model of Liquid Benchmarks

own datasets or defining their own tasks for running specially defined experiment in a *private* area which is separated from the public setup of the benchmarks.

Each liquid benchmark is configured by defining the following main components:

– *Scenarios:* In principle, each liquid benchmark consists of at least one scenario which describes a use case that focus on evaluating some aspects of the competing implementations in the target domain (e.g. MacroBenchmark or MicroBenchmark). Each scenario is described by the following main items:

  • Datasets: Each scenario should contain at least one dataset. These datasets can be of different types and different formats (e.g. image files, database records, XML files) depending on the context of the benchmark.

  - Tasks: Describes a set of operations that need to be executed by the competing implementations over the defined datasets (e.g. queries, update operations, compressing operations). Each operation represents one or more target evaluation aspect which is in the scope of the benchmark.
  - Metrics: Represents the measures of evaluating the performance of the competing implementation in executing the different tasks (e.g. execution time, response time, throughput, precision, recall). It provides the basis of comparing the competing implementations.
- *Evaluated Solutions:* The set of competing implementations (e.g. algorithms, techniques, systems) that solve the domain specific problem of the liquid benchmarks. It should be noted that each solution can have different *versions*. Each of these versions will be treated as a separate (but linked) competing solution. Each solution need to register the set of its supported tasks in order to avoid the running of many failing tasks.
- *Testing Environments:* Represents a set of different configuration of computing resources (e.g. operating system, main memory, processor, disk space) that reflect different real-world scenarios.

Based on the configuration of the components of the liquid benchmarks, users can run their *experiments* where each experiment is specified by: 1) The *task* to be executed with the selected metrics for evaluation. 2) The *solution(s)* (implementation(s)) to be evaluated. 3) The testing *environment* which will be used for running the experiment. Executing each experiment produces a set of *experimental results* which usually can be rendered by different *rendering styles* (e.g. csv, XML, HTML, Plot). The experimental results will also be subject to social contributions from the end-users in the form of *social text* (e.g. tagging, comments, discussion, blogging). The results of all experiments associated with their provenance information (e.g. time, user, execution parameters) can be stored in a centralized repository which is then used for performing different search and analysis operations. It is usual that some of the evaluated solutions is not able to execute *all* of the defined tasks of the benchmark scenario due to several reason (e.g. out of the main focus, under development). Therefore, each solution needs to register the set of its supported tasks in order to avoid running many failing experiments.

5 Liquid Benchmarks: Architecture and Implementation

In this section, we present the architecture for the Liquid Benchmarks platform, illustrated in Figure 2, that consists of two main parts:

1. *Benchmark Collaborative Development* part where benchmark developers produce an inventory of assets that can be shared with the community. For example, they can collaboratively define the benchmark scenarios, setup the competing solutions and configure the testing environments.
2. *Benchmark Usage* part where end-users can define, run and view the results of their experiments in addition to searching the results of previously

running experiments. Moreover, they can collaborate and act on the assets and knowledge of the benchmark to produce value, recognize undiscovered knowledge and create new knowledge.

The two parts of the benchmark architecture are equipped with several *components* that reflect the objectives of our approach. The roles of these components are described as follows:

- **Web-based User Interface:** Provides the end user with a user-friendly interface where he/she can specify the configuration of his experiment by making a choice of *evaluated solutions*, benchmarking *tasks* with its associated *dataset*, interesting *metrics* and *rendering style* of the results in addition the target *testing environment*. For all of these choices, the metadata is information is available through a **Metadata Store** component that captures all the required information during the benchmark design process. Moreover, the Liquid Benchmark UI provides the end-users with other features such as: managing user accounts, searching the results of previous experiments, submitting new experiments, uploading private datasets and defining private user-defined testing tasks.
- **Experiment Manager:** It receives the specification of the user-defined experiment which is configured by the Liquid Benchmark GUI which is then passed to the **Experiment Queue** for execution in the target *testing environment*. The experiment queue plays an important role in ensuring that the execution of one experiment in a testing environment does not influence the execution of another experiment in the same environment. Therefore, any experiment need to be waiting in the queue until its target testing environment becomes available. After the end of executing the experiment, the experiment manager stores the experiment results in a **Central Repository** and, in parallel, passes them with the *rendering style* information for the **Rendering Manager** component which presents the results for the end-user. Since the execution of some experiments can be too long, the experiment manager is responsible for stopping any experiment that exceed the maximum runtime. This maximum runtime represents a defined threshold where the value of this threshold can depend on many factors such as: the configuration of the target test environment, the task nature and the user priority. In addition, the experiment manager can limit the number of running experiments for any user in order to avoid overloading the system.
- **Repository of Experiment Results:** It stores the results of all previously running experiments with their associated configuration parameters, *provenance* information (e.g. timestamp, user) and social information (e.g. comments, discussions). Clearly, end-users can access and view the contents of this repository to explore the results of previously running experiments without taking the time of running any new ones.
- **Cloud-Based Computing Environments:** It hosts *testing environments* which are shared by the liquid benchmark end-users. In fact, the hosting environments should have variant and scaling (in terms of computing resources) configuration settings (e.g. CPU speed, disk storage, main memory) that

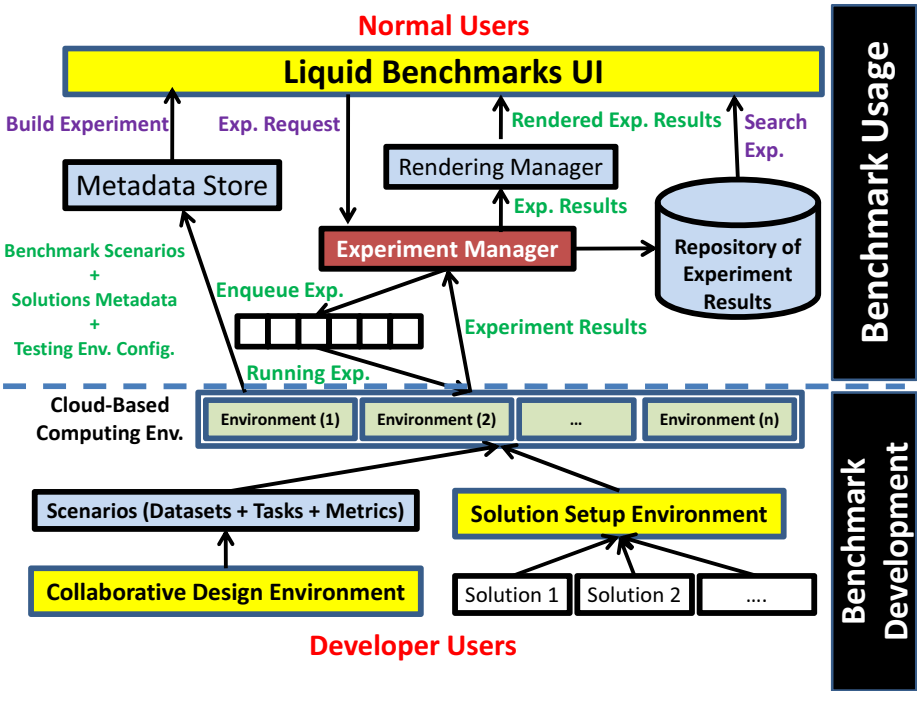

Fig. 2. Liquid Benchmarks: System Architecture

reflect different real-world scenarios. The rationale behind this is to ensure that the result of comparing the evaluated solutions is fair, comprehensive and insightful.

- **Collaborative Design Environment:** It is used by the set of registered developer users who have the privilege to build the specification of the benchmark scenarios (e.g. datasets, tasks, metrics). This environment provides the developers with the required tools to achieve their tasks (e.g. forums, wikis, shared file systems).
- **Solution Setup Environment:** This is another collaborative environment which is used by the developers to setup and configure the *competing solutions* in the different *testing environments*. After the configuration step, the solutions are running as accessible *services* for the end-users.

It is our ongoing work to realize the deployment of 3 case studies for liquid benchmarks by employing this architecture. In these case studies, we are using the *Amazon Web Services* (AWS) for establishing shared computing resources (e.g. Elastic Compute Cloud (EC2[8]), Simple Storage Service (S3[9]), Relational

[8] http://aws.amazon.com/ec2/
[9] https://s3.amazonaws.com/

Database Service (RDS[10])) and the open source social network platform, *elgg*[11]. These case studies are targeting the following domains:

1. *XML compression:* where we are reusing our previous work on benchmarking XML compression tools in the context of the *XMLCompBench* project [1,19]. We are currently deploying the resources of this benchmark to be available online for end-users where they can create, run and share their experiments. The benchmark provides an extensive experimental platform for benchmarking 9 of the available XML compression tools. The XML corpus of this benchmark consists of 57 documents which are covering the different types and scales of XML documents. The experiments are evaluating the XML compressors with 3 different metrics: compression ratio, compression time and decompression time. In order to provide the chance of assessing the consistency of the performance behaviors of the evaluated XML compressors, we are configuring two testing environments: one with high computing resource and the other with limited computing resources.
2. *SPARQL query processing:* where we are extending our previous work on benchmarking relational techniques for processing SPARQL queries [15,20]. In this case study, we are using the SPARQL Performance Benchmark (SP^2Bench) that have been recently proposed by Schmidt et al. [21] in order to compare between many of the available implementations (relational and non-relational) of SPARQL query processors (e.g. RDF-3X[12], AllegroGraph[13], Jena[14], Sesame[15], Virtuoso[16]). The performance characteristics of these implementations will be evaluated by different metrics such as: query execution time, data loading time, disk storage cost, memory consumption and scalability limit of handling very large RDF datasets.
3. *Graph query processing:* the field of graph databases and graph query processing has recently received a lot of attention due to the constantly increasing usage of graph data structure for representing data in different domains such as: chemical compounds, multimedia databases, social networks and biological pathways. Several techniques have been proposed to tackle the performance challenges of different graph query types [6]. However, so far, there is no standard benchmark or an experimental study to evaluate these approaches. Therefore, we are furnishing this case study by the available research and open source implementations in this domain (e.g. Closure-Tree [12], gIndex [24], TreePi [25], neo4j[17]). In addition, we are collaboratively trying with some experienced researchers to design an initial standard benchmark that can be used by the researchers in this domain.

[10] http://aws.amazon.com/rds/
[11] http://elgg.org/
[12] http://www.mpi-inf.mpg.de/~neumann/rdf3x/
[13] http://www.franz.com/agraph/allegrograph/
[14] http://jena.sourceforge.net/
[15] http://www.openrdf.org/
[16] http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/VOSRDF
[17] http://neo4j.org/

6 Conclusions and Outlook

The Web has dramatically enhanced the people's ability to share ideas, knowledge and contributions. We believe that the Computer Science research community should have the leadership in having such *scientific* collaborative environments that will significantly improve the ability of the scientific communities to understand their research problems, have clean and careful analysis for the state-of-the-art and gain insights that can help them to develop new effective technologies and solutions.

In this paper, we coined the notion of *liquid benchmark* as a first step towards an effective solution that rely on current advances in the Web technologies to provide *collaborative* Web-based platforms that facilitates the key tasks of evaluating, comparing and analyzing the *continuous* scientific contributions in different domains. We believe that our proposed solution can make the best use of the increasing human power participating in the Computer Science research efforts which are distributed over the world. In particular, we argue that our solution can empower the Computer Science research communities with many capabilities such as:

- Providing *workable* environments to collaboratively build standard benchmarks that can be widely used for achieving insightful evaluation for alternative research efforts. These environments can help researchers to optimize their time in assessing and improving the quality of their contribution. Having such environments will discourage authors from publishing paper with adhoc or poor experimental results.
- Developing centralized and *focused* repositories for related software implementations [2] and their experimental results. These repositories can be used as a very positive step towards finding the solutions for *repeatability* problems.
- Facilitating *collaborative* maintenance of experimental studies to guarantee their *freshness*. This task can follow the same model of collaborative organization of international conferences or journals where each participating researchers or research groups in a specific community can play a volunteering managerial role for a specific period.
- Facilitate the establishing of shared computing resources environment that can be utilized by different active contributors in the same domain who reside in different parts of the world.
- Leveraging the *wisdom of the crowd* in providing feedbacks over the experimental results in a way that can give useful insights for solving further problems and improving the state-of-the-art.
- Establishing a *transparent* platform for scientific *crediting* process based on collaborative community work.
- Creating concrete foundations and feasible environments for providing *provenance* services [17] for scientific experimental results and time-analysis service for the evolution of research efforts.

We recognize that our work is at a preliminary stage and may leave out some of the important details (e.g. credit attribution, data anonymization). However, we

hope that our proposal will serve as the foundation for a fundamental rethinking of the experimental evaluation process in the Computer Science field.

References

1. Benchmark of XML compression tools, http://xmlcompbench.sourceforge.net/
2. SourceForge: A free repository of open source software, http://sourceforge.net/
3. Transaction Processing Performance Council, http://www.tpc.org/default.asp
4. XMark Benchmark, http://monetdb.cwi.nl/xml/
5. Abadi, D.J., Marcus, A., Madden, S., Hollenbach, K.J.: Scalable Semantic Web Data Management Using Vertical Partitioning. In: VLDB, pp. 411–422 (2007)
6. Aggarwal, C.C., Wang, H. (eds.): Managing and Mining Graph Data. Springer, Heidelberg (2010)
7. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A., Stoica, I., Zaharia, M.: Above the Clouds: A Berkeley View of Cloud Computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley (February 2009)
8. Barbosa, D., Manolescu, I., Yu, J.X.: Microbenchmark. In: Encyclopedia of Database Systems, p. 1737 (2009)
9. Carey, M.J., DeWitt, D.J., Naughton, J.F.: The oo7 Benchmark. In: SIGMOD, pp. 12–21 (1993)
10. Crolotte, A.: Issues in Benchmark Metric Selection. In: Nambiar, R., Poess, M. (eds.) TPCTC 2009. LNCS, vol. 5895, pp. 146–152. Springer, Heidelberg (2009)
11. Dolog, P., Krötzsch, M., Schaffert, S., Vrandecic, D.: Social Web and Knowledge Management. In: Weaving Services and People on the World Wide Web, pp. 217–227 (2008)
12. He, H., Singh, A.K.: Closure-Tree: An Index Structure for Graph Queries. In: ICDE, p. 38 (2006)
13. Huppler, K.: The Art of Building a Good Benchmark. In: Nambiar, R., Poess, M. (eds.) TPCTC 2009. LNCS, vol. 5895, pp. 18–30. Springer, Heidelberg (2009)
14. La, H.J., Kim, S.D.: A Systematic Process for Developing High Quality SaaS Cloud Services. In: Jaatun, M.G., Zhao, G., Rong, C. (eds.) CloudCom 2009. LNCS, vol. 5931, pp. 278–289. Springer, Heidelberg (2009)
15. MahmoudiNasab, H., Sakr, S.: An Experimental Evaluation of Relational RDF Storage and Querying Techniques. In: Proceedings of the 2nd International Workshop on Benchmarking of XML and Semantic Web Applications (BenchmarX 2010), DASFAA Workshops (2010)
16. Manolescu, I., Afanasiev, L., Arion, A., Dittrich, J., Manegold, S., Polyzotis, N., Schnaitter, K., Senellart, P., Zoupanos, S., Shasha, D.: The repeatability experiment of sigmod 2008. SIGMOD Record 37(1), 39–45 (2008)
17. Miles, S., Groth, P.T., Deelman, E., Vahi, K., Mehta, G., Moreau, L.: Provenance: The Bridge Between Experiments and Data. Computing in Science and Engineering 10(3), 38–46 (2008)
18. Pavlo, A., Paulson, E., Rasin, A., Abadi, D.J., DeWitt, D.J., Madden, S., Stonebraker, M.: A comparison of approaches to large-scale data analysis. In: SIGMOD, pp. 165–178 (2009)
19. Sakr, S.: XML compression techniques: A survey and comparison. J. Comput. Syst. Sci. 75(5), 303–322 (2009)

20. Sakr, S., Al-Naymat, G.: Relational Processing of RDF Queries: A Survey. SIGMOD Record (2009)
21. Schmidt, M., Hornung, T., Lausen, G., Pinkel, C.: SP^2Bench: A SPARQL Performance Benchmark. In: ICDE, pp. 222–233 (2009)
22. Sidirourgos, L., Goncalves, R., Kersten, M.L., Nes, N., Manegold, S.: Column-store support for RDF data management: not all swans are white. PVLDB 1(2), 1553–1563 (2008)
23. Stonebraker, M.: A new direction for TPC? In: Nambiar, R., Poess, M. (eds.) TPCTC 2009. LNCS, vol. 5895, pp. 11–17. Springer, Heidelberg (2009)
24. Yan, X., Yu, P.S., Han, J.: Graph Indexing: A Frequent Structure-based Approach. In: SIGMOD, pp. 335–346 (2004)
25. Zhang, S., Hu, M., Yang, J.: TreePi: A Novel Graph Indexing Method. In: ICDE, pp. 966–975 (2007)

A Discussion on the Design of Graph Database Benchmarks*

David Dominguez-Sal[1], Norbert Martinez-Bazan[1], Victor Muntes-Mulero[1], Pere Baleta[2], and Josep Lluis Larriba-Pey[1]

[1] DAMA-UPC, Barcelona, Spain
{ddomings,nmartine,vmuntes,larri}@ac.upc.edu
[2] Sparsity Technologies, Barcelona, Spain
pbaleta@spartisty-technologies.com

Abstract. Graph Database Management systems (GDBs) are gaining popularity. They are used to analyze huge graph datasets that are naturally appearing in many application areas to model interrelated data. The objective of this paper is to raise a new topic of discussion in the benchmarking community and allow practitioners having a set of basic guidelines for GDB benchmarking. We strongly believe that GDBs will become an important player in the market field of data analysis, and with that, their performance and capabilities will also become important. For this reason, we discuss those aspects that are important from our perspective, i.e. the characteristics of the graphs to be included in the benchmark, the characteristics of the queries that are important in graph analysis applications and the evaluation workbench.

1 Introduction

The analysis and storage of data in the form of a graph has increased in the recent years. Analyzing the characteristics of social networks, the use of the Internet, or the interactions among proteins has put graph processing in the eye of the storm. The amount of data managed in most of those cases is huge, and the complexity of the algorithms needed for the analysis as well, leading to a clear need in the market: the Graph Database Management System (GDB).

A GDB is a step forward in the management and analysis of data. As stated by Angles and Gutierrez [1]: *"Graph database models can be defined as those in which data structures for the schema and instances are modeled as graphs or generalizations of them, and data manipulation is expressed by graph-oriented operations and type constructors*". Graph databases emphasize the queries that compute results related to the structure of the links in the graphs rather than on the entities themselves: for example detecting link patterns, path analysis, authority relations, etc. However, managing large graphs is a complex issue, and obtaining the best suited analysis algorithms is difficult.

* The members of DAMA-UPC thank the Ministry of Science and Innovation of Spain and Generalitat de Catalunya, for grant numbers TIN2009-14560-C03-03 and GRC-1087 respectively.

R. Nambiar and M. Poess (Eds.): TPCTC 2010, LNCS 6417, pp. 25–40, 2011.

There are a certainly growing number of initiatives to implement and commercialize GDBs, like Neo4j [2], HyperGraphDB [3], Infogrid [4] or DEX [5] and many RDF solutions such as Jena [6] or AllegroGraph [7]. There are other initiatives to create graph querying languages that allow for a simplified view of querying to the user like SPARQL [8] and Gremlin [9]. This shows that the community is very active proposing new technologies, and sets an adequate scenario to reflect on which is the adequate benchmark for a GDB.

The main objective of this paper is to open the discussion on GDB benchmarking. Thus, we describe and discuss important aspects to be considered for benchmarking. We started describing and analyzing the type of applications where it is necessary the use of GDBs. In particular, we review the application of GDBs in Social Network Analysis (SNA), proteomics, recommendation systems, travel planning and routing, which gives a catalog of representative areas where huge graph datasets are appearing.

We believe that the set of applications mentioned are representative of the marketplace for GDBs. Thus, based on those, we discuss the characteristics of the graphs that appear in such applications and how they could influence benchmarking. We also survey different types of operations, why they are important and how they can be categorized in order to produce a wide coverage of issues within a benchmark. Finally, we discuss on the evaluation setup of the benchmark, where issues like the experimental process to follow and the type of measures to be taken are considered. Despite the diversity of applications, we find that the different areas have common features, and we believe that the design of a benchmark based on SNA would become a representative candidate of general GDB applications.

The paper is organized as follows. We start by setting up the state of the art in Section 2. Then, in Section 3, we analyze a broad spectrum of applications demanding for massive graph management. From these scenarios, we extract generic characteristics of graphs and queries, that will be important in order to design a graph database benchmark. Also, we propose a query categorization and we remark the relevant aspects that we should take into account for the experimental settings of a benchmark. Finally, we draw some conclusions.

2 Graph Oriented Benchmarks

Popular database benchmarks, such as TPC-C or TPC-H [10], focus on evaluating relational database queries that are typical of a business application. These benchmarks emphasize queries with joins, projections, selections, aggregations and sorting operations. However, since GDBs aim at different types of queries, these widespread benchmarks are not adequate for evaluating their performance.

Object oriented databases (OODB) share some similarities with GDBs. The data of a OODB also conforms a graph structure, where the entities that are represented as objects draw relationships among them. The OO1 benchmark [11], one of the earliest proposals, is a very simple benchmark that emphasizes three basic operations for OODB: (a) lookup, which finds the set of objects for a given object identifier; (b) traversal, which performs a 7-hop operation starting from a

random node; and (c) insertion, which adds a set of objects and relations to the database. OO1 defines a dataset that only contains one type of objects with a fixed number of outgoing edges per object. Since the links mostly go to objects with a similar document identifier, the graphs are very regular.

Another popular benchmark for OODB is the OO7 proposed by Carey et al [12]. In OO7, the database contains three types of objects, which are organized as a tree of depth seven. The connectivity of the database is also very regular because objects have a fixed number of relations. The benchmark is made up by a rich set of queries that can be clustered into two groups: (a) traversal queries, which scan one type of objects and then access the nodes connected to them in the tree, and (b) general queries, which mainly perform selections of objects according to certain characteristics.

We observe that although OODB benchmarks create graphs, the graphs have a very different structure from typical graphs in graph analysis applications. As we review in detail in Section 3.2, graphs in GDBs are very irregular: the degree of the nodes exhibit a large variance, nodes are clustered in communities and graphs have small diameters. Furthermore, the applications that interact with GDBs are mainly interested in analyzing the graph structure, i.e. the relationships, instead of the attributes in the objects. For example, operations such as finding the shortest path connecting two objects or finding patterns (e.g. a clique) are common GDB operations that are not considered in OODB.

XML databases also follow a model which relates entities. An XML database is a collection of typed data and attributes organized as a tree. One of the most well known benchmarks for XML databases is XMARK [13], which models an auction site. The queries in the benchmark cover many aspects: selections, sorted access, tree path location, aggregation, etc. Nevertheless, XML only models trees, which are a limited subclass of graphs.

In the recent years, the knowledge management community has made efforts to design a standard for representing the relations between metadata elements, which has derived in the introduction of the Resource Description Framework (RDF). Along with RDF, the community has designed a query language called SPARQL, which is an SQL extension to describe relationship patterns among entities. In order to test the performance of these knowledge bases, Guo et al. proposed a benchmark of 14 SPARQL queries in [14], which is known as LUBM.

To our knowledge, the only benchmark proposed for the evaluation of graph libraries is the HPC Scalable Graph Analysis Benchmark v1.0 [15]. The benchmark is compound by four separated operations on a graph that follows a power law distribution: (a) insert the graph database as a bulk load; (b) retrieve the set of edges with maximum weight; (c) perform a k-hops operation; and (d) calculate the betweenness centrality of a graph, whose performance is measured as the number of edges traversed per second. However, this benchmark does not evaluate some features expected from a GDB such as object labeling or attribute management. For an implementation and a discussion of the benchmark over different graph databases see [16].

3 Benchmark Considerations

In this section, we discuss several aspects that may affect the design of a GDB benchmark. First, we examine a set of motivating scenarios where graph databases are useful. Second, we describe the most commonly used graph types. From this, we explore common queries run in these scenarios and propose a categorization depending on their interaction with the graph data. Finally, we review some experimental considerations for the design of GDB benchmarks.

3.1 Analysis of Motivating Applications

Over the past years, there has been an increasing interest in multiple disciplines on datasets that can be represented and analyzed as a network or graph. In these networks, the nodes represent the entities and the edges the interaction or relationships between them. For example, the use of Social Network Analysis (SNA) has grown to currently become one of the most successful tools to investigate the structure of organizations and social groups, focusing on uncovering the structure of the interactions between people [17]. SNA techniques have been effectively used in several areas of interest like social interaction and network evolution analysis, counter-terrorism and covert networks, or even viral marketing. The Web 2.0 [18] and the increasing use of Internet applications that facilitate interactive collaboration and information sharing has caused the appearance of many social networks of different kinds, like Facebook and LinkedIn for social interaction, or Flickr for multimedia sharing. Other web portals that contain human interactions can also be considered social networks, like in bibliographic catalogs such as Scopus, ACM or IEEE, where the scientific community is sharing information and establishing de facto relationships. In all these cases, there is an increasing interest in the analysis of the underlying networks, to obtain a better knowledge of the patterns and the topological properties. This may be used to improve service to users or even to provide more profit to the information providers in the form of direct advertising or personalized services.

The rapid growth of the World Wide Web, has caused new graph structured data to be archived and analyzed, such as hypertext and semi-structured data [19]. Also, with RDF, users are allowed to explicitly describe semantic resources in the form of graphs [20]. In this context, and others such as finding web service's connection patterns, algorithms and applications for graph and subgraph matching in data graphs are becoming increasingly important [21]. Pattern matching algorithms are also used to find relationships in social networks [22], find research collaboration partners, or mine the connections among research paper publications in archived bibliography datasets.

The use of graph theory combined with computing analysis has attracted the interest of the graph mining community [23]. Some of the classical analysis in this area are the determination of an actor's centrality, to identify key players, leaders or relevant people in a community; the grouping of individuals in communities or affinity groups, to provide specific services or to improve their connectivity; the identification of weaknesses or singular points in the network, for security

or counter-terrorism; or the study of social roles and collaborations, to get the position of an actor in the society or to connect people to others to improve their relationships network in professional environments.

Another area of interest is in proteomics or genetic interactions, where the large-scale study of proteins is considered the next step in the study of biological systems. The fact that most proteins act in collaboration with other proteins is the basis for proteomics to reveal which of those are potentially associated with a certain disease. In protein-protein interaction networks, nodes represent proteins, and edges between nodes represent physical interactions between proteins. Several protein interaction databases are shared between hundreds of research laboratories and companies around the world, such as BioGRID [24] or the Protein Data Bank (PDB) [25]. With this data, some of the most usual analysis in genetic interactions are the study of the complexity of the network topology, the node analysis of centrality and role, the identification of articulation points or bridges between groups of proteins, or the clustering of proteins based on their relationship or neighborhoods. Many scientific software tools are available for the analysis and the visualization of data, like Navigator [26] for network analysis, visualization and graphing.

Network analysis has also become essential for recommendation systems. In this case, the goal is to present information that could be interesting to the users based on previous knowledge extracted from their social environment. Recommendation systems are prior to computers, but the Internet has exploded again the use of recommendation for different purposes, such as on-line sales and catalogs like Amazon, or digital music services in iTune. Even PageRank [27] in Google can be considered a recommendation engine, and the analysis of hubs and authorities to rate Web pages in HITS [28] is an exploration of the network of relationships of a web page with its hyperlinks. Although relational databases have been the storage system of choice in many commercial recommendation engines for collaborative filtering, like Strands [29] for social networks and eCommerce, lately, new approaches have appeared using graph representation and exploration like conceptual graphs [30] or more recently DirectedEdge [31]. Another area of network analysis where graphs may be large is travel planning and routing, where the system tries to find the most efficient path between two points according to some constraints or recommendations given by the user, like in Amadeus [32], or real time analysis of traffic networks in large cities. In these cases, the data is naturally represented as graphs where vertices stand for the location and the edges are the routes with lengths and costs. Then, queries are mainly navigational operations between neighbors or network topology analysis.

3.2 Graph Description

GDBs store entities as nodes, which have relations among them that are set as edges. However, depending on the particular application, the graph model may differ, showing a different degree of complexity. In addition to the plain storage of nodes and edges, we detail the main features required by some applications:

Attributes: In addition to nodes and edges, graph databases store information associated to these nodes and edges. The associated information is typically string or numerical values, which indicate the features of the entity or relation. For the particular case of edges, some graphs include numerical attributes that quantify the relation, which is usually interpreted as the length, weight, cost or intensity of the relation. Moreover, many applications set a unique identifier for each node and edge of the graph that labels each graph element.

Directed: Depending on the problems the relation between two nodes may be symmetric or not. If the relation is symmetric, the edge might be traversed from any of the adjacent nodes to the opposite one. If the relation is not symmetric, edges differentiate the head and the tail. The tail of the edge is the node from which the edge starts, and the head of the edge is the node which the edge points to. Undirected graphs are a particular case of directed graphs, since an undirected edge can be represented as two directed edges, each one in a reverse direction of the other.

Node and edge labeling: Some applications differentiate different labels (or types) of nodes and edges. Labeling has an important impact because some applications require distinguishing between different kinds of relations. For example, a social network may accept either "positive" or "negative" friendship relations [33].

Multigraphs: Multigraphs differ from graphs in that two nodes can be connected by multiple edges. Multigraphs appear commonly when graphs have typed edges because often two nodes are related by different categories. For example, in a mobile telephone network that represents the cell phones as the nodes and the telephone calls as the edges, two nodes will have multiple connections if they have called more than once.

Hypergraphs: A hypergraph is a generalization of the concept of a graph, in which the edges are substituted by hyperedges. In contrast to regular edges, an hyperedge connects an arbitrary number of nodes. Hypergraphs are used, for example, for building artificial intelligence models [34].

Graph Characterization: Real graphs are typically very different from graphs following the Erdös-Renyi model (random graphs) [35]. Leskovec et al. analyzed over 100 real-world networks in [36] in the following fields: social networks, information/citation networks, collaboration networks, web graphs, Internet networks, bipartite networks, biological networks, low dimensional networks, actor networks, and product-purchaser networks. The size of these networks varies from a few hundred nodes to millions of nodes, and from hundreds of edges to more than one hundred million. We note that although they might seem huge, the graph data sets of some current applications are significantly larger: for example Flickr accounts more than 4 billion photographs that can be tagged and rated [37], and Facebook is publishing more than 25 billion pieces of content each month. For these large graphs, one of the most interesting aspects is that in general most graph metrics (such as the node degree or the edge weight) follow power law distributions [36,38,39], and hence some areas of the graph are significantly denser than others.

With respect to the characterization of graphs, we summarize some properties that often appear in these huge graphs [23]: (a) they contain a large connected component that accounts for a significant percentage of the nodes; (b) they are sparse, which means that the number of edges is far away from the maximum number of edges; (c) the degree distribution follows a power law distribution (i.e scale-free networks), where a few nodes have a number of connections that greatly exceeds the average, usually known as hubs; (d) the average diameter of each connected components from the graph is small, in other words, from a given node there is a short path to reach the majority of the remaining nodes in the connected component (which is also called the small world property); and (e) the nodes are grouped in communities where the density of edges among the members of the community is larger than the edges going outside the community.

Discussion: In this section, we see that graph applications represent their datasets following graphs with different degrees of complexity. Nevertheless, we observe that the structure of the graphs datasets follow power law characterizations and properties, which makes it possible to create generic graphs representative of multiple applications for benchmarking purposes.

According to the previously described applications, we also identify three aspects that a generic GDB should be able to compute (and thus be included in a benchmark): (a) labeled graphs, which enable to identify the nodes and edges of the graph; (b) directed graphs, which set the relation and its direction; (c) attribute support, which are used by applications such as for setting the weight of the edges.

3.3 Graph Operations

In this subsection, we present several types of operations used in the areas presented before. The analysis of this section will be useful to learn different aspects that will be used to fix criteria, in the following subsection, to design relevant operations for a future benchmark. Table 1 lists some of these graph operations, organized by the type of access that is performed on the graph.

First, we define a set of generic operations. These operations are not typical in a single specific domain, but common operations that may be necessary in any context. This set of operations allows us to (i) get atomic information from the graph such as *getting a node*, *getting the value of an attribute of an edge*, or *getting the neighbor nodes of a specific node*, and (ii) *create*, *delete* and *transform* any graph. Any complex query or transformation of the graph will necessarily use these operations.

Afterwards, we extend these operations to other higher level actions typically performed in the scenarios presented before. We group these operations into different types:

Traversals. Traversals are operations that start from a single node and explore recursively the neighborhood until a final condition, such as the depth or visiting a target node, is reached. For instance, we consider the operation of calculating a *shortest path*, which is the shortest sequence of edges (or the smallest addition

Table 1. Graph Operations, Areas of Interest and Categorization

Group	Operation	Social Network	Protein Interaction	Recom-mendation	Routing	Analytical	Cascaded	Scale	Attr.	Result
Generic operations										
General Atomic / Local Information Extraction	Get node/edge	+	+	+	+	Yes	No	Neigh.	No	Set
	Get attribute of node/edge	+	+	+	+	Yes	No	Neigh.	No	Set
	Get neighborhood	+	+	+	+	Yes	No	Neigh.	No	Set
	Node degree	+	+	+	+	Yes	No	Neigh.	No	Agr.
General Atomic Transformations	Add/Delete node/edge	+	+	+	+	No	No	Neigh.	No	Set
	Add/Delete/Update attrib.	+	+	+	+	No	No	Neigh.	E/N	Set
Application dependent operations										
Traversals	(Constrained) Shortest Path	+	+		+	Yes	Yes	Glob.	Edge	Graph
	k-hops	+		+	+	Yes	Yes	G/N	No	Graph
Graph Analysis	Hop-Plot	+				Yes	No	Glob.	No	Agr.
	Diameter	+	+			Yes	Yes	Glob.	Edge	Set
	Eccentricity	+				Yes	Yes	Glob.	Edge	Agr.
	Density	+	+			Yes	No	Glob.	No	Agr.
	Clustering coefficient	+				Yes	Yes	Glob.	No	Agr.
Components	Connected Components	+	+			Yes	Yes	Glob.	No	Graph
	Bridges	+	+		+	Yes	Yes	Glob.	No	Set
	Cohesion	+				Yes	Yes	Glob.	No	Set
Communities	Dendrogram	+				Yes	Yes	Glob.	No	Graph
	Max-flow min-cut	+				Yes	Yes	Glob.	Edge	Graph
	Clustering	+	+	+		Yes	Yes	Glob.	No	Graph
Centrality Measures	Degree Centrality	+		+		Yes	No	Glob.	No	Set
	Closeness Centrality	+		+		Yes	Yes	Glob.	No	Set
	Betweenness Centrality	+		+		Yes	Yes	Glob.	No	Set
Pattern Matching	Graph/Subgraph Matching	+	+			Yes	Yes	Neigh.	No	Graph
Graph Anonymization	k-degree Anonym.	+		+		Yes	No	Glob.	No	Graph
	k-neighborhood Anonym.	+		+		Yes	Yes	Glob.	No	Graph
Other Operations (Similarity, ranking,...)	Structural Equivalence	+				Yes	Yes	Glob.	No	Graph
	PageRank			+		Yes	No	Glob.	Node	Set

of edge weights in the case of weighted graphs) that connects two nodes. In a directed graph the direction is restricted to outgoing edges from the tail to the head. Note that shortest paths may be constrained by the value of some node or edge attributes, as in the case of finding the shortest route from two points, avoiding a certain type of road, for instance. This operation is also used as a measure to calculate the information loss of graph anonymization methods. Another typical operations is calculating *k-hops*, which returns all the nodes that are at a distance of k edges from the root node. A particular case is when $k = 1$, also known as the *neighbors* of the node. The particular case of *1-hops* is widely used as part of other operations. For example to calculate the *nearest neighborhood* in recommender systems, to obtain a particular user's neighborhood with similar interest, or in web *ranking* using hubs and authorities.

Graph Analysis. Basic graph analysis includes the study of the topology of graphs to analyze their complexity and to characterize graph objects. It is basically used to verify some specific data distributions, to evaluate a potential match of a specific pattern, or to get detailed information of the role of nodes and edges. In several situations graph analysis is the first step of the analytical process and it is widely used in SNA and protein interaction analysis. Among this we may calculate the *hop-plot* (a metric to measure the rate of increase of the neighborhood depending on the distance to a source node), the (*effective*) *diameter*, the *density*, or the *clustering coefficient* (to measure the degree of transitivity of the graph), to give some examples.

Components. A *connected component* is a subset of the nodes of the graph where there exists a path between any pair of nodes. Thus, a node only belongs to a single connected component of the graph. Finding connected components is usually crucial in many operations, typically used in a preprocess phase. Also, some operations are helpful to study the vulnerability of a graph, or the probability to separate a connected component into two other components. Finding *bridges*, edges whose removal would imply separating a connected component, is important in many applications. Going further, the *cohesion* of the graph can be computed by finding the minimum number of nodes that disconnect the group if removed.

Communities. A community is generally considered to be a set of nodes where each node is closer to the other nodes within the community than to nodes outside it. This effect has been found in many real-world graphs, especially social networks. Operations related to the creation of a community may be building *dendograms* (communities formed through hierarchical clustering), finding the *minimal-cut* set of edges or other *clustering* techniques.

Centrality Measures. A centrality measure aims at giving a rough indication of the social importance of a node based on how well this node connects the network. The most well-known centrality measures are *degree*, *closeness* and *betweenness centrality*.

Pattern matching. Pattern recognition deals with algorithms which aim at recognizing or describing input patterns. *Graph matchings* are usually categorized into exact or approximate. Exact matchings may include finding homomorphisms

or (subgraph) isomorphisms. Approximate matchings may include error-correcting (subgraph) isomorphisms, distance-based matching, etc.

Graph Anonymization. The anonymization process generates a new graph with properties similar to the original one, avoiding potential intruders to reidentify nodes or edges. This problem gets more complex when the nodes and edges contain attributes and the problem goes beyond the anonymization of the pure graph structure. The anonymization of graphs becomes important when several actors exchange datasets that include personal information. To give a couple of examples, two anonymization procedures are the the k-degree anonymity of vertices, or the k-neighborhood anonymity, which guarantees that each node must have k others with the same (one step) neighborhood characteristics.

Other operations. There are other operations related to the applications presented in Subsection 3.1. For instance, finding similarity between nodes in a graph has shown to be very important in SNA. An example of this is *structural equivalence*, which refers to the extent to which nodes have a common set of linkages to other nodes in the system. Also, specially for recommendation systems, ranking the nodes of a graph is an important issue (for instance *PageRank*).

Discussion: We observe that over a small set of generic operations that are shared by all scenarios, applications compute a rich set of more specific graph operations. Moreover, according to Table 1, SNA provides one of the most rich set of graph operations, which makes social networks a candidate scenario for designing benchmarks that are representative of applications that use GDBs.

3.4 Query Categorization

The computational requirements of graph queries is not homogeneous. For example, some queries may traverse the full graph, while others may request the outdegree of a single node. In order to build a balanced benchmark it must be representative of the different types of operations that can be issued by an application to the graph database. In this section, we build up a set of categories to classify the different operations that are issued to a graph database:

Transformation/Analysis: We distinguish between two types of operations to access the database: transformations and analysis operations. The first group comprise operations that alter the graph database: bulk loads of a graph, add/remove nodes or edges to the graphs, create new types of nodes/edges/attributes or modify the value of an attribute. The rest of queries are considered analysis queries. Although an analysis query does not modify the graph, it may need to access to secondary storage because the graph or the temporary results generated during the query resolution are too large to fit in memory.

Cascaded access: We differentiate two access patterns to the graph: cascaded and not cascaded. We say that an operation follows a cascaded pattern if the query performs neighbor operations with a depth at least two. For example, a 3-hops operation follows a cascaded pattern. Thus, a non cascaded operation may access a node, an edge or the neighbours of a node. Besides, an operation that does not request the neighbours of a node, though it may access the full graph, is a non cascaded operation. For instance, an operation that returns the node with the largest

value of an attribute accesses all nodes, but since it does not follow the graph structure is a non-cascaded operation.

Scale: We classify the queries depending on the number of nodes accessed. We distinguish two types of queries: global and neighbourhood queries. The former type corresponds to queries that access the complete graph structure. In other words, we consider as global queries those that access to all the nodes or the edges of the graph. The latter queries only access to a portion of the graph.

Attributes: Graph databases do not only have to manage the structural information of the graph, but also the data associated to the entities of the graph. Here, we classify the queries according to the attribute set that it accesses: edge attribute set, node attribute set, mixed attribute set or no attributes accessed.

Result: We differentiate three different types of results: graphs, aggregated results, and sets. The most typical output for a graph database query is another graph, which is ordinarily a transformation, a selection or a projection of the original graph, which includes nodes and edges. An example of this type of result is getting the minimum spanning tree of a graph, or finding the minimum length path that connects two nodes. The second type of results build up aggregates, whose most common application is to summarize properties of the graph. For instance, a histogram of the degree distribution of the nodes, or a histogram of the community size are computed as aggregations. Finally, a set is an output that contains either atomic entities or result sets that are not structured as graphs. For example, the selection of one node of a graph or finding the edges with the greatest weight are set results.

Discussion: Queries in a benchmark must represent the workload of the real environment where the application is going to be executed, and thus should be adapted to the particular application profile to be tested. We have seen that graph operations are diverse, but many operations share similar operational patterns. In Table 1, we summarize these patterns categorizing the catalog of popular operations. In this table, we find that although the most basic operations are neither structured nor affect large fractions of the graph, many applications use large scale operations that traverse the graph. Furthermore, we find that most graph operations are accessing the information stored in the edges since the attributes in the edges (and weights in particular) are modeling the relations between entities, which is the main objective of a graph. We also observe that generic GDB must be able to store temporal objects because they are necessary for most operations (e.g: storing a boolean to decide whether a node has been visited or counting the number of paths through a node). Finally, we see that generic GDB must be able to manage different types of result sets because we find operations in many different applications that return sets, aggregates and graphs. We note that although most operations analyze the graph, many applications may store the results as attributes. In other words, although the results of a complex operation such as community detection are analytical, they may be stored as attributes that may be updated periodically.

In general, operations with a complexity larger than linear (over the number of nodes or edges) should be included with care in a GDB benchmark because they

may become unfeasible to compute for large graphs. If these operations are very representative of a certain workload, then one possible approach is to accept approximate results. For example, the betweenness centrality measure is very appreciated in the analysis of social networks, but in practice it is seldom computed with an exact algorithm because of its high computational costs [40]. Therefore, if large graphs are considered, benchmarks may also consider approximated implementations though metrics about quality of the result and precise descriptions of the approximated algorithm are recommended.

3.5 Experimental Setting

Experimental setup and measurement is one of the most important parts of a benchmark and it must be clearly defined and configured to allow a fair comparison between multiple solutions in different environments. Early database benchmarks only focused in the response time of the queries. As the benchmarks have evolved and platforms are more complex, the measurements have become more sophisticated and, in consequence, the experimental setup and the testing process is more expensive and complicated. For example, early benchmarks like Wisconsin only considered the response time using very simple scale factors [41]. Later, TPC-C or TPC-H introduced a metric with the relationship of the pricing with respect the maximum throughput [42], and more recently LUBM [14], for the benchmarking of RDF graphs, defined a combined metric between the query response time and the answer completeness and soundness in the case of partial results and pattern matching.

The different concepts related to the experimental setup and configuration of a graph database benchmark can be grouped in the following areas: configuration and setup, experimental process, and measures.

Configuration and setup: Modern benchmarks allow for the definition of the dataset size by means of a *scale factor*, which fixes the dataset size generated following precise data distributions. For graphs, the scale factor defines the number of nodes and edges. Additionally, the graph must conform to scale the graph structural properties for different scales such as the vertex degree and hop-plot distributions, the diameter or the community structure [43].

A second set of parameters defines the allowed capabilities to preprocess the dataset. Some of the most important factors to define are: (a) *data partitioning*, which usually allows for a flexible graph partitioning using either horizontal partitioning, (each structure partitioned), vertical partitioning (structures are grouped following some criteria), or hybrid; (b) *indexing*, that allows the GDB to build freely indexes that speedup queries at the expense of slower loads; (c) *redundancy*, that specifies if data may be replicated; and if (d) *data reorganization* procedures that optimize the internal data structures of the GDB are allowed. In order to have precise measures, benchmarks usually distinguish between two parts: (a) a load phase where the data is preprocessed, and (b) a query phase where the ingested data cannot be reindexed or reorganized.

Finally, another aspect is the definition of the data consistency requirements that the database must support during the benchmark. In transactional

benchmarks, ACID properties are usually the standard. However, since many graph operations are analytical, more relaxed approaches can be taken, like eventual consistency or simply requiring concurrence for multiple readers.

Experimental process: The experimental process definition minimizes the side-effects produced by the way the experiments are executed, or by some influences that may appear between experiments, like the the processor caches or the execution order of the queries. (a) The *warm-up* sets the state of the computer before the benchmark is executed. It usually allows to populate the memory with a fraction of the graph (or indexes) as a result of some warm up queries. On the other hand, if the benchmarks aims at measuring the I/O overhead, it is necessary to ensure that the operating system's cache is emptied. (b) The *query generation* sets in which order the queries are executed (e.g. in sequential order). For non sequential benchmarks, it also sets the probability distribution that incoming queries fit. (c) The *observational sampling procedure* describes the number of executions for each query and how they are summarized. The observations are collected once a condition happens, which is typically once a query (or set of queries) finishes or a period of time lapses. Optionally, outliers or the fastest and slowest results might be discarded. It should be taken into consideration that when data is modified by a query it is also important to include into the measure the flush of the graph updates to disk.

Measures: Measures are closely related to the benchmark capabilities and to the audience. Some general measures that can be applied to graphs are the following: (a) the *load time*, which measures the elapsed time for loading and preprocess the dataset; (b) the *size* of the graph; (c) the *query response time* that accounts for the time elapsed between the query is issued until the results are output; (d) the *throughput* that measures the number of queries completed in an interval of time; (e) the *price* of the computing site including hardware, license and maintenance costs if applicable; or (f) the *power consumption* that measures the energy requirements of the computing equipment and gives an indirect measure of its cooling requirements. In order to compare such metrics among different platforms, it is common to introduce normalized metrics such as the *price/throughput* or the *power/throughput* that enable an easier comparison of benchmarking results between, for example, a supercomputer and a laptop hardware setup.

In the area of graph benchmarking, some specialized metrics have been proposed. The HPC benchmark [15] defines the *number of traversals per second* (TEPS) as the average number of edges explored during a traversal per second. The TEPS gives a measure of the average effort to navigate through the relationships with respect to the size of the graph. For pattern matching, when it is difficult to find all the results in a deterministic way like in knowledge bases, Guo et al. have proposed to measure the *query completeness* (or recall) or how far is the result respect to all the possible answers [14]. If the GDB is able to return results as soon as they appear, it might be interesting to obtain a plot of the GDB's recall with respect to time.

Discussion: There are many experimental settings and measures that can be selected for a GDB benchmark. The final selection depends on the benchmark characteristics, the goals and the audience. Thus, an industrial-oriented benchmark will probably focus on the throughput and cost of complex queries on very large graphs. However, research oriented benchmarking may be interested in the performance of more specific operations for different hardware configurations. In contrast to other scenarios where ACID properties are mandatory, we believe that many applications of GDB benchmarks allow for more relaxed consistency behaviors.

4 Conclusions

In this paper, we have analyzed important aspects for the design of a GDB benchmark. First of all, there is a significant core of applications that benefit significantly from their management as a graph, like social network analysis, protein interaction, recommendation and routing among others. These applications justify by themselves the existence and evolution of GDBs, and at the same time, justify the existence and evolution of a GDB benchmark. Its correct design and implementation implies the following aspects to be considered:

- The characteristics of the graph to be used in a benchmark are important. Considering the inclusion of directed graphs with attributes, with different node and edge types in the context of multigraphs would be important.
- The characteristics of the graph like the distribution of edges per node, attribute values per edge, etc. depend on the application and should be applied based on the different studies appeared in the literature. Nevertheless, most huge graph datasets follow power law distributions.
- Although not necessarily all the operations appearing in our analysis need to be considered for a benchmark, both analytical and transformation operations should be present.
- The cascaded nature of many graph operations is important, and a benchmark should include a good selection of operations with and without this cascaded nature.
- While there are operations that cover a traversal of all the database, others just affect a few of their components. Such feature should be evaluated taking into consideration the metrics to be used, in order to balance the importance of each case.
- Depending on the application, some operations just evaluate the structure of the graph, while others take the attributes in the nodes and specially in the edges, to be evaluated. A good combination of queries with both characteristics would be of interest for the proper deployment of a benchmark.
- The nature of the result is important because GDBs are capable of offering a good assortment of answers. In particular, operations returning sets of nodes, graphs or aggregational answers would be recommended for a balanced benchmark.
- There are other aspects that influence the fairness of a benchmark. Those are the configuration of the GDB (partitioning, indexing of attributes, data redundancy and reorganization), the way the experimental process is undertaken

(warm-up of the database, query generation and the observational procedure) and the metrics to be considered (load time, repository size, query response time, throughput obtained and the cost of the deployment). However, those aspects are not totally influenced by the GDB environment.

Just to finalize, it would be very important to find an application covering as many of the aspects that we have evaluated in this paper as possible. We believe that social network analysis is very significant because it covers almost all the operation types in graph databases, it is easy to understand by the final user and carries a lot of natural queries with answers that can be conceptually understandable. Any other application with such characteristics would be of use and beneficial for a GDB benchmark.

References

1. Angles, R., Gutiérrez, C.: Survey of graph database models. ACM Comput. Surv. 40(1) (2008)
2. Neo4j: The neo database (2006), http://dist.neo4j.org/neo-technology-introduction.pdf
3. HypergraphDB: HypergraphDB website, http://www.kobrix.com/hgdb.jsp (last retrieved in March 2010)
4. Infogrid: Blog, http://infogrid.org/blog/2010/03/operations-on-a-graph-database-part-4 (last retrieved in March 2010)
5. Martínez-Bazan, N., Muntés-Mulero, V., et al.: Dex: high-performance exploration on large graphs for information retrieval. In: CIKM, pp. 573–582 (2007)
6. Jena-RDF: Jena documentation, http://jena.sourceforge.net/documentation.html (last retrieved in March 2010)
7. AllegroGraph: AllegroGraph website, http://www.franz.com/agraph/ (last retrieved in May 2010)
8. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. W3C (2008), http://www.w3.org/TR/rdf-sparql-query/
9. Gremlin website: Gremlin documentation, http://wiki.github.com/tinkerpop/gremlin/ (last retrieved in June 2010)
10. Transaction Processing Performance Council (TPC): TPC Benchmark. TPC website, http://www.tpc.org (last retrieved in June 2010)
11. Cattell, R., Skeen, J.: Object operations benchmark. TODS 17(1), 1–31 (1992)
12. Carey, M., DeWitt, D., Naughton, J.: The oo7 benchmark. In: SIGMOD Conference, pp. 12–21 (1993)
13. Schmidt, A., Waas, F., Kersten, M., Carey, M., Manolescu, I., Busse, R.: Xmark: A benchmark for xml data management. In: VLDB, pp. 974–985 (2002)
14. Guo, Y., Pan, Z., Heflin, J.: Lubm: A benchmark for owl knowledge base systems. J. Web Sem. 3(2-3), 158–182 (2005)
15. Bader, D., Feo, J., Gilbert, J., Kepner, J., Koetser, D., Loh, E., Madduri, K., Mann, B., Meuse, T., Robinson, E.: HPC Scalable Graph Analysis Benchmark v1.0. HPC Graph Analysis (February 2009)
16. Dominguez-Sal, D., Urbón-Bayes, P., Giménez-Vañó, A., Gómez-Villamor, S., Martínez-Bazán, N., Larriba-Pey, J.L.: Survey of graph database performance on the hpc scalable graph analysis benchmark. In: Shen, H.T., Pei, J., Özsu, M.T., Zou, L., Lu, J., Ling, T.-W., Yu, G., Zhuang, Y., Shao, J. (eds.) WAIM 2010. LNCS, vol. 6185, pp. 37–48. Springer, Heidelberg (2010)

17. INSNA: International network for social network analysis, http://www.insna.org/
18. OReilly, T.: What is Web 2.0: Design patterns and business models for the next generation of software (2005)
19. Abiteboul, S., Buneman, P., Suciu, D.: Data on the Web: from relations to semistructured data and XML. Morgan Kaufmann Publishers Inc., San Francisco (2000)
20. Brickley, D., Guha, R.V.: Resource description framework (rdf) schema specification 1.0. W3C Candidate Recommendation (2000)
21. Shasha, D., Wang, J., Giugno, R.: Algorithmics and applications of tree and graph searching. In: PODS, pp. 39–52. ACM, New York (2002)
22. Anyanwu, K., Sheth, A.: ρ-queries: Enabling querying for semantic associations on the semantic web. In: WWW, pp. 690–699. ACM Press, New York (2003)
23. Chakrabarti, D., Faloutsos, C.: Graph mining: Laws, generators, and algorithms. ACM Computing Surveys (CSUR) 38(1), 2 (2006)
24. BioGRID: General repository for interaction datasets, http://www.thebiogrid.org/
25. PDB: Rcsb protein data bank, http://www.rcsb.org/
26. NAViGaTOR, http://ophid.utoronto.ca/navigator/
27. Brin, S., Page, L.: The anatomy of a large-scale hypertextual Web search engine. Computer Networks and ISDN Systems 30(1-7), 107–117 (1998)
28. Kleinberg, J.M.: Authoritative sources in a hyperlinked environment. J. ACM 46(5), 604–632 (1999)
29. Strands: e-commerce recommendation engine, http://recommender.strands.com/
30. Chein, M., Mugnier, M.: Conceptual graphs: fundamental notions. Revue d'Intelligence Artificielle 6, 365–406 (1992)
31. DirectedEdge: a recommendation engine, http://www.directededge.com (last retrieved in June 2010)
32. Amadeus: Global travel distribution system, http://www.amadeus.net/
33. Leskovec, J., Huttenlocher, D., Kleinberg, J.: Signed networks in social media. In: CHI, pp. 1361–1370 (2010)
34. Goertzel, B.: OpenCog Prime: Design for a Thinking Machine. Online wikibook (2008), http://opencog.org/wiki/OpenCogPrime
35. Erdos, P., Renyi, A.: On random graphs. Mathematicae 6(290-297), 156 (1959)
36. Leskovec, J., Lang, L., Dasgupta, A., Mahoney, M.: Statistical properties of community structure in large social and information networks. In: WWW, pp. 695–704 (2008)
37. Flickr: Four Billion, http://blog.flickr.net/en/2009/10/12/4000000000/ (last retrieved in June 2010)
38. Faloutsos, M., Faloutsos, P., Faloutsos, C.: On power-law relationships of the internet topology. In: SIGCOMM, pp. 251–262 (1999)
39. McGlohon, M., Akoglu, L., Faloutsos, C.: Weighted graphs and disconnected components: patterns and a generator. In: KDD, pp. 524–532 (2008)
40. Bader, D., Madduri, K.: Parallel algorithms for evaluating centrality indices in real-world networks. In: ICPP, pp. 539–550 (2006)
41. Bitton, D., DeWitt, D., Turbyfill, C.: Benchmarking database systems a systematic approach. In: VLDB, pp. 8–19 (1983)
42. Transaction Processing Performance Council (TPC): TPC Benchmark H (2.11). TPC website, http://www.tpc.org/tpch/ (last retrieved in June 2010)
43. Leskovec, J., Chakrabarti, D., Kleinberg, J., Faloutsos, C., Ghahramani, Z.: Kronecker graphs: An approach to modeling networks. Journal of Machine Learning Research 11, 985–1042 (2010)

A Data Generator for Cloud-Scale Benchmarking

Tilmann Rabl, Michael Frank, Hatem Mousselly Sergieh, and Harald Kosch

Chair of Distributed Information Systems
University of Passau,
Germany
{rabl,frank,moussell,kosch}@fim.uni-passau.de
http://www.dimis.fim.uni-passau.de

Abstract. In many fields of research and business data sizes are breaking the petabyte barrier. This imposes new problems and research possibilities for the database community. Usually, data of this size is stored in large clusters or clouds. Although clouds have become very popular in recent years, there is only little work on benchmarking cloud applications. In this paper we present a data generator for cloud sized applications. Its architecture makes the data generator easy to extend and to configure. A key feature is the high degree of parallelism that allows linear scaling for arbitrary numbers of nodes. We show how distributions, relationships and dependencies in data can be computed in parallel with linear speed up.

1 Introduction

Cloud computing has become an active field of research in recent years. The continuous growth of data sizes, which is already beyond petabyte scale for many applications, poses new challenges for the research community. Processing large data sets demands a higher degree of automation and adaptability than smaller data sets. For clusters of thousand and more nodes hardware failures happen on a regular basis. Therefore, tolerance of node failures is mandatory. In [19] we sketched a benchmark for measuring adaptability, in this paper we present a data generator that is cloud aware, as it is designed with the top goals of cloud computing, namely scalability and decoupling, i.e. avoidance of any interaction of nodes [3].

Traditional benchmarks are not sufficient for cloud computing, since they fall short on testing cloud specific challenges [2]. Currently, there are only a few benchmarks available specifically for cloud computing. The first one was probably the TeraSort Benchmark[1]. Others followed, such as MalStone and CloudStone. These benchmarks are dedicated to a single common task in cloud computing. While this kind of benchmarks is essential for scientific research and evaluation, it fails to give a holistic view of the system under test. We think that there is a need for a benchmark suite that covers various aspects of cloud computing. The database

[1] The current version can be found at http://www.sortbenchmark.org/

R. Nambiar and M. Poess (Eds.): TPCTC 2010, LNCS 6417, pp. 41–56, 2011.

community has traditionally an affection for simple benchmarks [11,22]. Although reduction of complexity is a basic principle of computer science and unnecessary complexity should be avoided by all means, there seems to be a trend to simplistic evaluations. In order to get meaningful results benchmarks should have diverse and relevant workloads and data [5]. Often it is best to use real life data. For example, in scientific database research the Sloan Digital Sky Survey is frequently referenced [24]. Yet for many applications such as social websites there is no data publicly available. And even though storage prices are dropping rapidly, they are still considerably high for petabyte scale systems. A current 2 TB hard drive costs about USD 100, resulting in a GB price of about USD 0.05. So the price for 1 PB of raw disk space is about USD 50000. Therefore, it is not sensible to store petabytes of data only for testing purposes. Besides storage, the network necessary to move petabytes of data in a timely manner is costly. Hence, the data should be created where it is needed. For a cluster of nodes this means that each node generates the data it will process later, e.g. load into a data base. In order to generate realistic data, references have to be considered, which usually requires reading already generated data. Examples for references are foreign keys. For clusters of computers this results in a fair amount of communication between nodes.

For example consider a table representing a many-to-many relationship between two tables. When generating corresponding keys one needs information about the keys in the two tables participating in the many-to-many relationship. On a single node system it is usually much faster to read in the two tables to create the relationship. But if the data for the two tables is scattered across multiple nodes, it is more efficient to regenerate it. This way the relationship can be created completely independently of the base tables. By using distributed parallel random number generators, such as the leap frog method [10, Ch.10], even the generation of single tables can be parallelized. Since the generation is deterministic, also references can still be computed independently.

In this paper we present an ubiquitous parallel data generation framework (PDGF) that is suitable for cloud scale data generation. It is highly parallel and very adaptable. The generator uses XML configuration files for data description and distribution. This simplifies the generation of different distributions of specified data sets. The implementation is focused on performance and extensibility. Therefore, generators for new domains can be easily derived from PDGF. The current set of generators is built to be nearly write-only, so they generate values only according to random numbers and relatively small dictionaries, but without rereading generated data. This is done to reduce I/O and network communication to the absolute minimum.

Consider the simplified example in Figure 1, taken from the eLearning management system presented in [19]. There are three tables, *user*, *seminar*, and *seminar_user*. The generation of tables *user* and *seminar* are straightforward. For *seminar_user* it has to be considered, that only *user_id*s and *seminar_id*s are generated, that actually exist in *user* and *seminar*. This is only easy if both attributes have continuous values, otherwise it has to be assured that the referenced tuples exist. A second challenge is that *degree_program* is replicated

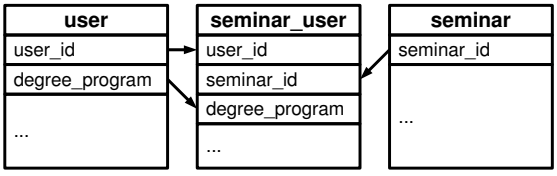

Fig. 1. Fragment of the schema of an eLearning management system

in *seminar_user*, so the combination of *user_id* and *degree_program* have to exist in *user*. Finally the values of tables like *seminar_user* have typically non uniform distributions.

The common solution to generate the table *seminar_user* is to first generate the two tables that are referenced and then use a look up or scan to generate the distribution. If this is done in parallel, either the referenced data has to be replicated, or the data generating process has to communicate with other nodes. This is feasible for smaller clusters, but for cloud scale configurations the communication part will be the bottleneck. Therefore, we propose a fully computational approach. Basically, our data generation is a set of functions that map a virtual row id to a tuple's attribute. Using this approach, we can easily recompute every value. So for the example above we would define a function for each attribute in the original tables. To generate uncorrelated data, the first computational step is usually either a permutation or a pseudo random number generation. For the example above this would only be needed for the *degree_program*. The value could either be chosen from a dictionary or be generated. To generate entries of *seminar_user*, two pseudo random numbers in the range of $[1, |user|]$ and $[1, |seminar|]$ are computed, with according distribution properties and then the function to generate *degree_program* is used, resulting in a valid tuple for *seminar_user*. This can be computed completely independently of the generation of *user* and *seminar*. Since parallel pseudo random number generators are used, *seminar_user* can be generated on any reasonable number of computing nodes in parallel.

This flexibility opens a broad field of application. Besides traditional relational, row oriented data, our system can easily generate data in other storage models, such as the Decomposed Storage Model [9], column wise as in MonetDB [6] or C-Store [23] or even mixed models [20].

An often discussed problem is that experiments are run on too small data sets [15]. This is not because of the negligence of researchers, but because large sized experiments are very time consuming and many researchers have no access to unlimited sized storage clusters. Hence, in many situations it seems sensible to use only a fraction of the data set and simulate a fraction of the system. Obviously, there are many research fields where such a methodology leads to realistic results. An example would be sort experiments. Our data generator is capable of generating statistically sound extracts of data sets as well as it can generate large data sets in uncorrelated fractions. So large scale test can either be scaled down or processed sequentially.

The rest of the paper is organized as follows, Section 2 gives an overview of previous work on data generation for database benchmarking. After that, Section 3 describes how the design goals of the data generator are met. Section 4 explains the data generation approach. The architecture of the data generator is described in Section 5. We evaluate the performance of our data generator in Section 6 and conclude with future work in Section 7.

2 Related Work

There has been quite some research on data generation for performance benchmarking purposes. An important milestone was the paper by Gray et al. [12], the authors showed how to generate data sets with different distributions and dense unique sequences in linear time and in parallel. Fast, parallel generation of data with special distribution characteristics is the foundation of our data generation approach.

According to their reference generation procedure, data generators can be roughly divided into three categories: no reference generation, scanning references, and computing references. No reference generation means that no relationships between tables are explicitly considered. So references are either only simple or based on mathematical probabilities. In this scheme it is for example not possible to generate foreign keys on a non-continuous unique key. Examples are data sets that only consists of single tables or data sets (e.g. SetQuery [17], TeraSort, MalGen [1], YCSB [8]) or unrelated tables (e.g. Wisconsin database [4], Bristlecone[2]).

Scanned references are generated reading the referenced tuple, this is either done simultaneously to the generation of the referenced tuple or by scanning the referenced table. This approach is very flexible, since it allows a broad range of dependencies between tables. However, the generation of dependent tables always requires the scanning or calculation of the referenced table. When the referenced table is read, additional I/Os are generated, which in many applications will limit the maximum data generation speed. Generating tables simultaneously does not constitute a problem. However, it requires generating all referenced tables. This is very inefficient, if the referenced tables are very large and don't need to be generated, e.g. for an materialized view with aggregation. Most systems that generate references use scanned references. An example is dbgen[3], the data generator provided by the TPC for the TPC-H benchmark [18]. Another approach was presented by Bruno and Chaudhuri [7], it largely relies on scanning a given database to generate various distributions and interdependencies. In [14] Houkjær et al. describe a graph based generation tool, that models dependencies in a graph and uses a depth-first traversal to generate dependent tables. A similar approach was presented by Lin et al. [16]. Two further tools that offer quite similar capabilities are MUDD [21] and PSDG [13]. Both feature description languages for the definition of the data layout and advanced distributions.

[2] Available at http://www.continuent.com/community/lab-projects/bristlecone
[3] dbgen can be downloaded from http://www.tpc.org/tpch/

Furthermore, both tools allow parallel generation. However, as described above the independent generation of dependent data sets is not possible.

A computed reference is recalculated using the fact, that the referenced data is deterministically generated. This results in a very flexible approach that also makes it possible to generate data with cyclic dependencies. The downside is the computational cost for regenerating the keys. However, as our test shows (see Section 6) current hardware is most often limited by I/O speed. To the best of our knowledge our tool is the only one that relies on this technique for parallel data generation.

3 Design Goals

PDGF's architecture was designed with the following goals: platform independence, extensibility, configurability, scalability and high performance. The following sections explain how each goal is met.

3.1 Platform Independence

To achieve a high degree of platform independence, PDGF was written in Java. It has been tested under Windows and different Linux distributions. Using Java has no degrading effect on the performance of the generator. In fact, startup and initialization times can be neglected compared to the time taken to generate terabytes of data for the cloud. This characteristic gives the just-in-time compilation technology enough time to compile and optimize the code, so that a similar generation speed as with pre-compiled binaries can be achieved.

3.2 Extensibility

PDGF is a framework that follows a black box plug-in approach. PDGF components can be extended and exchanged easily. Custom plug-ins can be implemented without consideration of programming aspects like parallelism or internal details. Furthermore, some minor built-in components are also exchangeable via the plug-in approach. These components include the file caching strategies and the scheduling strategy which are responsible for work splitting among threads and nodes. To make the system aware of new plug-ins it is sufficient to place them in the classpath and to reference them in database scheme description file.

3.3 Configurability

PDGF can be configured by two XML-based configuration files. One file configures the runtime environment, while the other configures the data schema and generation routines.

Runtime Configuration File. The runtime configuration file determines which part of the data is generated on a node and how many threads are used. This

is used for splitting the work between participating nodes. The file is optional, since these settings can also be specified via command line or within the built-in mini shell. Listing 1 shows a sample runtime configuration file. This example is for Node 5 out of 10 nodes. Two worker threads are used.

```
<?xml version="1.0" encoding="UTF-8"?>
<nodeConfig>
 <nodeNumber>5</nodeNumber>
 <nodeCount>10</nodeCount>
 <workers>2</workers>
</nodeConfig>
```

Listing 1. "Runtime configuration file"

Data Schema Configuration File. The data schema configuration file is used to specify how the data is generated. It follows a hierarchical structure as illustrated below. As mentioned above, all major components (e.g. the random number generator, data generators, etc.) can be exchanged by plug-ins. To use a plug-in, its qualified class name has to be specified in the name attribute of the corresponding element (see listing 2).

```
<project name="simpleE-learning">
 <scaleFactor>1</scaleFactor>
 <seed>1234567890</seed>
 <rng name="DefaultRandomGenerator" />
 <output name="CSVRowOutput">
  <outputDir>/tmp</outputDir>
 </output>
 <tables>
  <table name="user">
   <size>13480</size>
   <fields>
    <field name="user_id">
     <type>java.sql.Types.INTEGER</type>
     <primary>true</primary> <unique>true</unique>
     <generator name="IdGenerator" />
    </field>
    <field name="degree_program">
     <type>java.sql.Types.VARCHAR</type>
     <size>20</size>
     <generator name="DictList">
      <file>dicts/degree.dict</file>
     </generator>
    </field>
   </fields>
  </table>
  <table name="seminar">  [..]  </table>
```

```
  <table name="user_seminare">
   <size>201754</size>
   <fields>
    <field name="user_id">
     <type>java.sql.Types.INTEGER</type>
     <reference>
      <referencedField>user_id</referencedField>
      <referencedTable>user</referencedTable>
     </reference>
     <generator name="DefaultReferenceGenerator">
      <distribution name="LogNormal">
       <mu>7.60021</mu> <sigma>1.40058</sigma>
      </distribution>
     </generator>
    </field>
    <field name="degree_program"> [..] </field>
    <field name="seminar_id">     [..] </field>
   </fields>
  </table>
 </tables>
</project>
```

Listing 2. "Config file example for userSeminar references"

3.4 Scalability

To work in cloud scale environments a data generator must have a deterministic output regardless the number of participating nodes. Moreover, it is necessary to achieve a nearly linear performance scaling with an increasing number of participating nodes.

To fulfill these requirements, we followed an approach that avoids the usage of a master node or inter-node and inter-process communication. Every node is able to act independently and process its share of workload without considering the data generated by other nodes. To achieve this kind of independence, a combination of multiple instances of random number generators with inexpensive skip-ahead functionality was applied. Allowing each node to determine its workload is done as follows: Each node starts a process and initializes it with the total number of participating nodes in addition to the node number. The workload is divided into work units. Each work unit contains the current table to be processed, the start row number and the end row number. For example, assuming that we have a table T1 with 1000 rows and 10 nodes (node1 to node10), then the workload is divided as follows:

node1 work unit 1..100
node2 work unit 101..200
..
node10 work unit 901..1000

The same approach is applied for every table of the database schema. Of course PDGF also utilizes modern multi-core CPUs by spawning a worker thread per (virtual) core. The number of worker threads is automatically determined or can be set manually in the configuration. After the workload has been distributed on the nodes, the work unit of a table is further divided and distributed on the worker threads of each node using the same approach. However, the start value of the work unit of the node should be added to the start and the end of each worker workload. For instance, assume that Node 2 has four worker threads. The work unit will be split among the worker threads.

node2 work unit 101..200
    worker1 work unit 101..125
    worker2 work unit 126..150
    worker3 work unit 151..175
    worker4 work unit 176..200

3.5 High Performance

PDGF achieves high performance by applying a set of strategies that allow efficient computation by minimizing the amount of I/O operations. To avoid synchronization overhead, custom per thread buffering and caching is used. Furthermore, the overhead caused by object creation was reduced by applying eager object initialization and the strict usage of shared reusable data transfer objects on a per thread basis.

Recalculating the value for a field on demand is usually less expensive than reading the data back into memory, as can be seen in the evaluation below. The static scheduler also generates no network traffic during the data generation. Avoiding read operations on the generated data limits I/O reads to the initialization stage. Only data generators that require large source files (i.e. dictionaries for text generation) produce I/O read operations. Therefore, an abstraction layer between the framework and the files was added to provide direct line access through an interface. This layer can easily be extended to use new file caching strategies. In the current implementation all files, independent of their size, are completely cached in the memory during the startup phase. With current hardware this is no problem for files up to hundreds of megabytes. There are many other aspects that directly or indirectly aid the performance, e.g. the efficient seeding strategy or the fast random number generator.

4 Data Generation Approach

To generate field values independently and to avoid costly disk reads, a random number generator (RNG) and a corresponding seed are assigned to each field. The used RNGs are organized hierarchically so that a deterministic seed for each field can be calculated, see Figure 2. To calculate a value for a field in a specific row the field's RNG only has to be re-seeded with the corresponding deterministic seed.

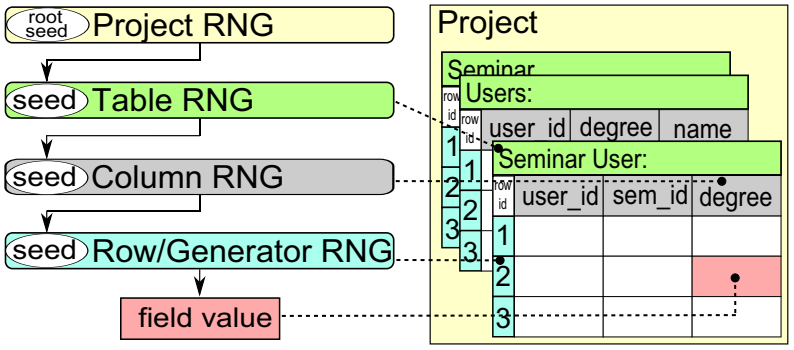

Fig. 2. Seeding strategy of PDGF

The RNG of the project element is considered as the root RNG and its seed is considered as the root seed. Each seed is used to derive the value of the seeds of the RNGs at the next lower level of the hierarchy. For example, the project seed is used to calculate the seed of the RNG of the tables and that of a table is used to generate seeds for columns and so on. Since there is only one project seed, all other seeds can be derived from that seed. As table and field (column) count are static, their seeds are cached after the first generation. Usually PDGF does not need to run through the complete seeding hierarchy to determine the seed for the generator RNG. It is sufficient to re-seed the field RNG with its pre-calculated seed, skip forward to the row needed and get the next value. For the default RNG used in PDGF this is a very inexpensive operation. After seeding, the RNG is passed to the corresponding field generator to generate the actual value.

5 Architecture

Figure 3 shows the architecture of the PDGF data generator. It consists of 5 basic components:

- The controller
- the view
- the generators, which contain
  - field generators
  - distribution functions
  - the random number generator
- The output module for generated data
- The scheduler

Controller/View. The controller takes user input such as the configuration files from the command line or a built in interactive mini-shell. This input is used to configure an instance of PDGF and to control the data generation process. By

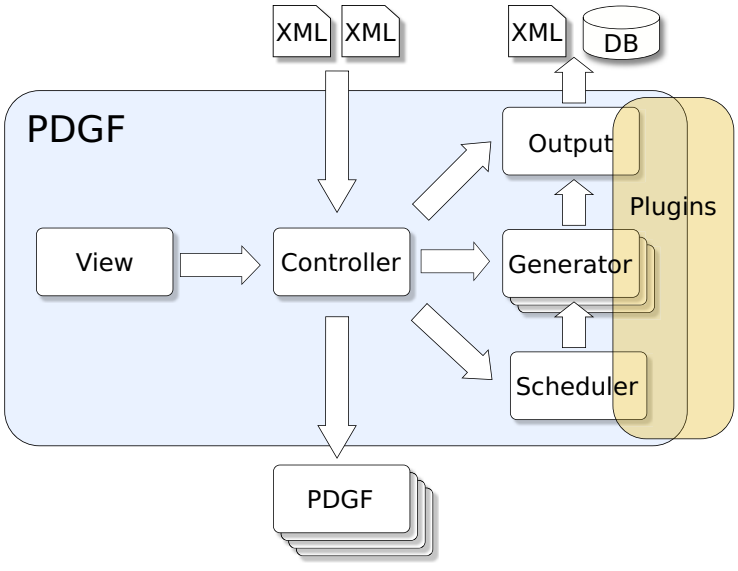

Fig. 3. Architecture of PDGF

default PDGF is a stand alone command line tool but other views can be easily attached to the controller e.g. a GUI. The controller allows the use of PDGF as a library. Distributed startup is currently realized by an external shell script, to allow the use with queuing systems. As PDGF can be used as a library, it is possible to use more complex startup algorithms.

Field Generators. The field generators are the most important part as they determine how the values for a field are generated. PDGF comes with a set of generators for general purpose: data, default reference, dictionary, double values, ids, int values, pseudo text grammar and static values. Since some data sets require a special structure, e.g. the TPC-H benchmark data set, PDGF provides a simple interface enabling easy implementation of generator plug-ins without problems with parallelization or concurrency.

Distribution Functions. The distribution functions allow generators to easily adapt and exchange how the values will be distributed. The distribution function uses the uniform random values from the RNG provided to each generator to calculate the non-uniformly distributed values. As for the field generators PDGF comes with some basic distribution functions: beta, binomial, exponential, log-normal, normal, Poisson, and Student's-t.

Random Number Generator. A parallel random number generator is the key to make the seeding algorithm efficient and fast. The used RNG generates random numbers by hashing successive counter values where the seed is the initial value for the counter. This approach makes skipping ahead very inexpensive. The

default RNG can also be exchanged and in addition it is even possible to specify a custom RNG per Generator. As for all other plug-ins, there is an interface that allows the usage of other RNGs.

Output Module. The output module determines how to save the generated data. An output module receives the values of an entire row for a table along with some meta information. By default the generated data is written to a comma separated value file, one per table and instance. Another possibility is to convert the generated values into SQL insert statements. These can either be written to a file or sent directly to a DBMS. In contrast to the other plug-in types an output plug-in is a serialization point of PDGF as all workers write concurrently to it.

Scheduler. The scheduler is the most complex plug-in. It influences the framework behavior to a large degree. The scheduler is responsible for dividing the work among physical nodes and the worker threads on the nodes. PDGF's default strategy is to statically divide the work between nodes and workers in junks of equal size. This is efficient if the data is generated in a homogeneous cluster or similar environment. In a heterogeneous environment the static approach leads to varying elapsed times among the participating nodes.

6 Performance Evaluation

We use the TPC-H and the SetQuery databases to evaluate our data generator [18,17]. All tests are conducted on a high performance cluster with 16 nodes. Each node has two Intel Xeon QuadCore processors with 2 GHz clock rate, 16 gigabyte RAM and two 74 GB SATA hard disks configured with RAID 0. Additionally, a master node is used, which has the same configuration, but an additional hard disk array with a capacity of 2 TBytes. For both benchmarks two test series are conducted. The first series tests the data generator's scalability in terms of data size on one node. The second series demonstrates the data generator's scalability in terms of the number of nodes with fixed data size. Each test was repeated at least five times. All results are averages of these test runs.

SetQuery. The SetQuery data set consists of a single table *BENCH* with 21 columns. 13 of the columns are integers with varying cardinalities from 2 to $1,000,000$ of which 12 are generated randomly. 8 additional columns contain strings, one with 8 characters and the others with 20 characters. The table size is scaled linearly according to a scaling factor SF, where $SF = 1$ results in about 220 MB. First we generated a 100 GB data set on 1 to 16 nodes (i.e. scaling factor 460). As can be seen in Figure 4 the average speed up per node is linear to the number of nodes. One node was able to generate about 105 MB per second. The super linear speed up for a higher number of nodes results from caching effects, which can be seen more clearly in the second test.

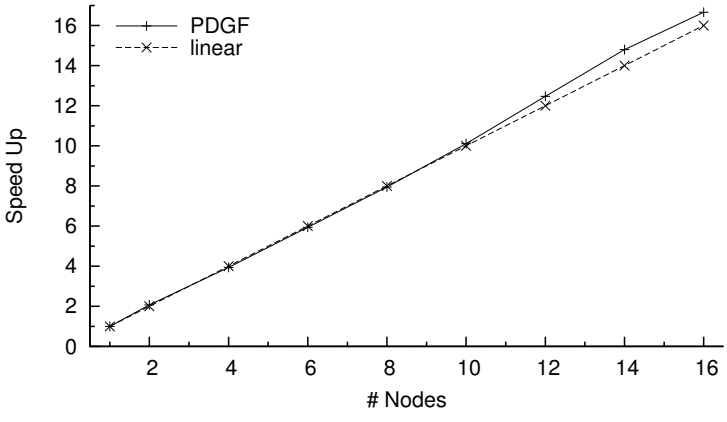

Fig. 4. Scaleup results for 1 to 16 nodes for a 100 GB SetQuery data set

For the second test, different data sizes are generated on a single node. We use scale factor 1 to 460. The resulting elapsed times for data generation are shown in Figure 5. It can be seen that the data generation scales well with the amount of data. However, the generation is not constant. This is due to caching effects and initialization. For smaller data sizes the initialization overhead decreases the overall generation speed. Then at scaling factor 10 (i.e. about 2 GB) there is a peak that results from hard disk and disk array caches. For larger data sizes the hard disks write speed is the bottleneck and limits the generation speed to about 100 MB/s.

TPC-H. To test a more complex scenario and compare the generation speed with other data generation tools, we used our data generator to generate TPC-H data. TPC-H defines 8 tables with different sizes and different number of columns. The schema defines foreign key relations and the following data types: integer; char; varchar; decimal and date. The amount of data is scaled linearly with the scale factor SF, where $SF = 1$ will result in 1 GB of data. Again, we tested the data generator's scalability in terms of the amount of data and the number of nodes. Figure 6 shows the data generation elapsed times for scale factor 1, 10 and 100 for a single node. Additionally, we generated the same data sizes with dbgen. Both axes of the figure are in logarithmic scale. To obtain fair results, dbgen was started with 8 processes, thus fully exploiting the 8 core system. Generation times for both tools were CPU bound. Since we had notable variations in the runtime of dbgen, we only report the best of 5 runs for each scaling factor. As can be seen in the figure, our configurable and extensible Java implemented tool can compete with a specialized C implementation.

Figure 7 shows the data generation elapsed times for different cluster sizes. For all cluster sizes the data generation elapsed time scales linearly with the data size. Furthermore, the generation time for certain scale factors decreases

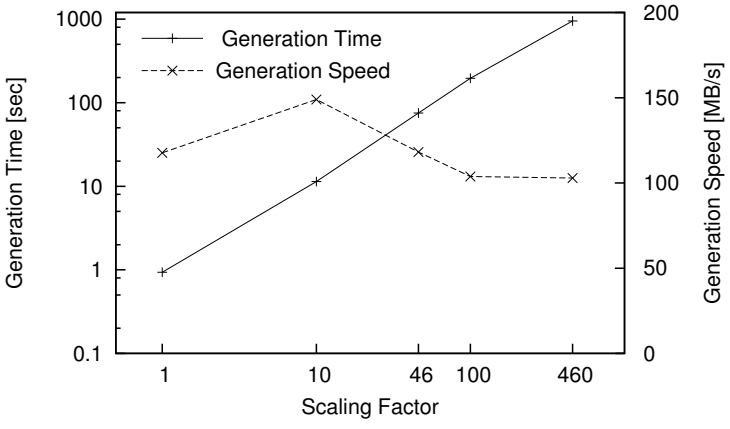

Fig. 5. Generation time and speed for different scaling factors of the SetQuery data set

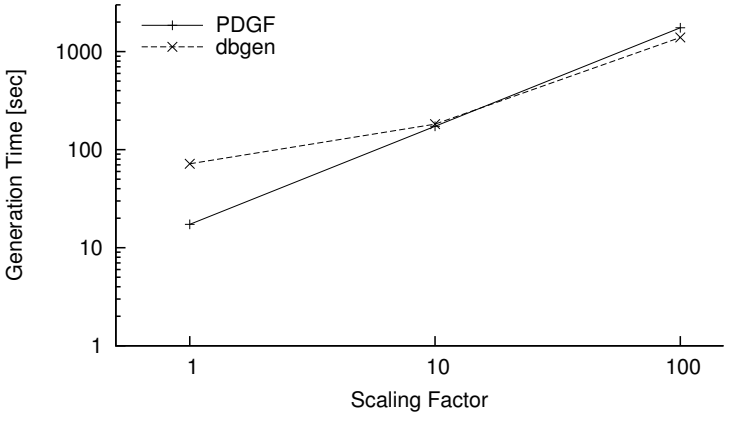

Fig. 6. Comparison of the generation speed of dbgen and PDGF

linearly with the number of nodes it is generated on. However, for scale factor 1 on 10 and 16 nodes the generation speed is significantly slower than for the other configurations. This is due to the large initialization overhead compared to the short generation time.

E-Learning. To measure data generation speed for more complicated distributed values we executed our simple example with the configuration file shown in 2. Even in this example with log normal distributed values and reference calculation, the generation speed is only limited by the hard disk speed. The values are therefore similar to the SetQuery results.

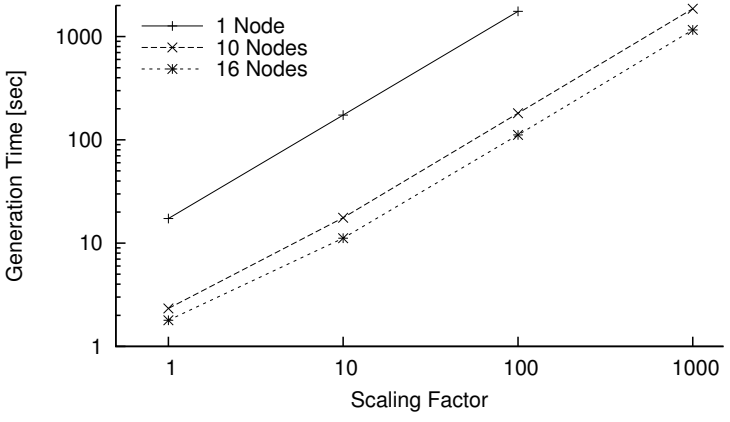

Fig. 7. Generation times of TPC-H data sets on different cluster sizes

7 Conclusion

In this paper we presented a framework for parallel data generation for benchmarking purposes. It uses XML files for the data definition and the configuration file. Like other advanced data generators (e.g. [21,7,13,14]) it features dependencies between relations and advanced distributions. However, it uses a new computational model, which is based on the fact that pseudo random numbers can be recomputed deterministically. Using parallel pseudo random number generators, dependencies in data can be efficiently solved by recomputing referenced data values. Our experiments show, that this model allows our generic, Java implemented data generator to compete with C implemented, specialized data generators.

For future work we are intending to further expand our set of generators and distributions. Furthermore, we will implement a GUI to allow a more convenient configuration. We also want to include other features, as for example schema and distribution retrieval from existing databases. To further increase the performance, we will include new schedulers that reduce wait times for slower nodes, as well as caching strategies to reduce re-computation of repeatedly used values.

To complete our benchmarking suite, we will use the data generator to implement a query generator. For this we will introduce time series generators. This will enable the generation of varying query streams as we presented in [19]. Furthermore, it will enable realistic time related data generation.

Acknowledgement

The authors would like to acknowledge Meikel Poess for his helpful comments and feedback on earlier versions of the paper.

References

1. Bennett, C., Grossman, R., Seidman, J.: Malstone: A benchmark for data intensive computing. Technical report, Open Cloud Consortium (2009)
2. Binnig, C., Kossmann, D., Kraska, T., Loesing, S.: How is the weather tomorrow?: Towards a benchmark for the cloud. In: Proceedings of the Second International Workshop on Testing Database Systems, DBTest 2009, pp. 1–6. ACM, New York (2009)
3. Birman, K., Chockler, G., van Renesse, R.: Toward a cloud computing research agenda. SIGACT News 40(2), 68–80 (2009)
4. Bitton, D., DeWitt, D.J., Turbyfill, C.: Benchmarking database systems: A systematic approach. In: Proceedings of the 9th International Conference on Very Large Data Bases, VLDB 1983, San Francisco, CA, USA, pp. 8–19. ACM, Morgan Kaufmann Publishers Inc. (November 1983)
5. Blackburn, S.M., McKinley, K.S., Garner, R., Hoffmann, C., Khan, A.M., Bentzur, R., Diwan, A., Feinberg, D., Frampton, D., Guyer, S.Z., Hirzel, M., Hosking, A.L., Jump, M., Lee, H., Moss, J.E.B., Phansalkar, A., Stefanovic, D., VanDrunen, T., von Dincklage, D., Wiedermann, B.: Wake up and smell the coffee: evaluation methodology for the 21st century. Communications of the ACM 51(8), 83–89 (2008)
6. Boncz, P.A., Manegold, S., Kersten, M.L.: Database architecture evolution: Mammals flourished long before dinosaurs became extinct. In: Proceedings of the 35th International Conference on Very Large Data Bases, VLDB 2009, pp. 1648–1653. VLDB Endowment (2009)
7. Bruno, N., Chaudhuri, S.: Flexible database generators. In: Proceedings of the 31st International Conference on Very Large Data Bases, VLDB 2005, pp. 1097–1107. VLDB Endowment (2005)
8. Cooper, B.F., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R.: Benchmarking cloud serving systems with ycsb. In: Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC 2010, pp. 143–154. ACM, New York (2010)
9. Copeland, G.P., Khoshafian, S.: A decomposition storage model. In: Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data, SIGMOD 1985, pp. 268–279. ACM, New York (1985)
10. Foster, I.: Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering. Addison Wesley, Reading (1995)
11. Gray, J.: Database and transaction processing performance handbook. In: Gray, J. (ed.) The Benchmark Handbook for Database and Transaction Systems, 2nd edn. Morgan Kaufmann Publishers, San Francisco (1993)
12. Gray, J., Sundaresan, P., Englert, S., Baclawski, K., Weinberger, P.J.: Quickly generating billion-record synthetic databases. In: Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, SIGMOD 1994, pp. 243–252. ACM, New York (1994)
13. Hoag, J.E., Thompson, C.W.: A parallel general-purpose synthetic data generator. SIGMOD Record 36(1), 19–24 (2007)
14. Houkjær, K., Torp, K., Wind, R.: Simple and realistic data generation. In: Proceedings of the 32nd International Conference on Very Large Data Bases, VLDB 2006, pp. 1243–1246. VLDB Endowment (2006)
15. Korth, H.F., Bernstein, P.A., Fernández, M.F., Gruenwald, L., Kolaitis, P.G., McKinley, K.S., Özsu, M.T.: Paper and proposal reviews: is the process flawed? SIGMOD Record 37(3), 36–39 (2008)

16. Lin, P.J., Samadi, B., Cipolone, A., Jeske, D.R., Cox, S., Rendón, C., Holt, D., Xiao, R.: Development of a synthetic data set generator for building and testing information discovery systems. In: Proceedings of the Third International Conference on Information Technology: New Generations, ITNG 2006, Washington, DC, USA, pp. 707–712. IEEE Computer Society, Los Alamitos (2006)
17. O'Neil, P.E.: The set query benchmark. In: Gray, J. (ed.) The Benchmark Handbook for Database and Transaction Systems, 2nd edn. Morgan Kaufmann Publishers, San Francisco (1993)
18. Poess, M., Floyd, C.: New tpc benchmarks for decision support and web commerce. SIGMOD Record 29(4), 64–71 (2000)
19. Rabl, T., Lang, A., Hackl, T., Sick, B., Kosch, H.: Generating shifting workloads to benchmark adaptability in relational database systems. In: Nambiar, R.O., Poess, M. (eds.) TPCTC 2009. LNCS, vol. 5895, pp. 116–131. Springer, Heidelberg (2009)
20. Ramamurthy, R., DeWitt, D.J., Su, Q.: A case for fractured mirrors. In: Proceedings of the 28th International Conference on Very Large Data Bases, VLDB 2002, pp. 430–441. VLDB Endowment (2002)
21. Stephens, J.M., Poess, M.: Mudd: a multi-dimensional data generator. In: Proceedings of the 4th International Workshop on Software and Performance, WOSP 2004, pp. 104–109. ACM, New York (2004)
22. Stonebraker, M.: A new direction for tpc? In: Nambiar, R.O., Poess, M. (eds.) TPCTC 2009. LNCS, vol. 5895, pp. 11–17. Springer, Heidelberg (2009)
23. Stonebraker, M., Abadi, D.J., Batkin, A., Chen, X., Cherniack, M., Ferreira, M., Lau, E., Lin, A., Madden, S., O'Neil, E.J., O'Neil, P.E., Rasin, A., Tran, N., Zdonik, S.B.: C-store: A column-oriented dbms. In: Proceedings of the 31st International Conference on Very Large Data Bases, VLDB 2005, pp. 553–564. VLDB Endowment (2005)
24. Szalay, A.S., Gray, J., Thakar, A., Kunszt, P.Z., Malik, T., Raddick, J., Stoughton, C., van den Berg, J.: The sdss skyserver: Public access to the sloan digital sky server data. In: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, SIGMOD 2002, pp. 570–581. ACM, New York (2002)

How to Advance TPC Benchmarks with Dependability Aspects

Raquel Almeida[1], Meikel Poess[2], Raghunath Nambiar[3], Indira Patil[4], and Marco Vieira[1]

[1] CISUC - Department of Informatics Engineering, University of Coimbra, Portugal
rrute@dei.uc.pt,
mvieira@dei.uc.pt
[2] Oracle Corporation, 500 Oracle Pkwy, Redwood Shores, CA 94065, USA
meikel.poess@oracle.com
[3] Cisco Systems, Inc., 3800 Zanker Road, San Jose, CA 95134, USA
rnambiar@cisco.com
[4] Hewlett Packard Company, 19410 Homestead Ave, Cupertino, CA 95014, USA
indira.patil@hp.com

Abstract. Transactional systems are the core of the information systems of most organizations. Although there is general acknowledgement that failures in these systems often entail significant impact both on the proceeds and reputation of companies, the benchmarks developed and managed by the Transaction Processing Performance Council (TPC) still maintain their focus on reporting bare performance. Each TPC benchmark has to pass a list of dependability-related tests (to verify ACID properties), but not all benchmarks require measuring their performances. While TPC-E measures the recovery time of some system failures, TPC-H and TPC-C only require functional correctness of such recovery. Consequently, systems used in TPC benchmarks are tuned mostly for performance. In this paper we argue that nowadays systems should be tuned for a more comprehensive suite of dependability tests, and that a dependability metric should be part of TPC benchmark publications. The paper discusses WHY and HOW this can be achieved. Two approaches are introduced and discussed: augmenting each TPC benchmark in a customized way, by extending each specification individually; and pursuing a more unified approach, defining a generic specification that could be adjoined to any TPC benchmark.

Keywords: Industry standard benchmarks, ACID properties, Durability, Dependability.

1 Introduction

Transaction Processing Performance Council (TPC) benchmarks [1] have proven useful to both information system vendors and purchasers. While vendors use the results of TPC benchmarks to demonstrate performance competitiveness for their existing products and to improve the performance of those that are under development, many purchasers require TPC benchmark results when considering new systems. During the past two decades the TPC has had a significant impact on the transactional systems and industry. However, although it raised the bar on expectations of industry standard

R. Nambiar and M. Poess (Eds.): TPCTC 2010, LNCS 6417, pp. 57–72, 2011.

benchmarks by providing long lasting relevant performance metrics and sound benchmarking methodologies, its benchmarks fall behind when measuring emerging quality attributes of modern transactional systems, such as high availability, reliability and maintainability.

Transactional systems are the core of the information systems of most organizations. Hence, organizations expect their systems to be highly dependable. Even though systems vendors have been continuously improving quality and reliability of hardware components, in this day and age dependability has a much broader scope. The term dependability has been established by the International Federation for Information Processing (IFIP) Working Group 10.4 [2]. It defines dependability as "*the trustworthiness of a computing system which allows reliance to be justifiably placed on the service it delivers*". In practice, dependability is an integrative concept that includes the following attributes [3], [4]:

- Availability (readiness for correct service)
- Reliability (continuity of correct service)
- Safety (absence of catastrophic consequences on user(s) and environment)
- Confidentiality (absence of unauthorized disclosure of information)
- Integrity (absence of improper system state alterations)
- Maintainability (ability to undergo repairs and modifications).

Although there is general acknowledgement that system failures often entail significant impact both on the proceeds and reputations of companies, measuring a system's ability to react and deal with such failures has been largely neglected by the TPC [14]. This is due to the fact that assessing dependability attributes of computer systems or computer system components dependents on many factors. Some of those factors are internal to systems, either pertaining to hardware or software, while others are external to systems, such as the environment or the user. In practice, assessing system dependability is a difficult problem that has been addressed by using both model-based and measurement-based techniques. The former include analytical [5] and simulation [6] models, and the latter include field measurement [7], fault injection [8],[9] and robustness testing [10],[11].

In TPC benchmarks, the well known Atomicity, Consistency, Isolation and Durability property tests (ACID) constitute a partial measure of the dependability of a system. First coined by Jim Gray in the late 1970s and later formally defined by Andreas Reuter and Theo Haerder, these important tests define reliable transactional systems. *Atomicity* requires the Database Management System (DBMS) to assure that any database modifications follow an 'all or nothing' rule. The DBMS must maintain the atomic nature of transactions in spite of any application, DBMS, operating system or hardware failure. *Consistency* ensures that the database remains in a consistent state, i.e. any transaction takes the database from one consistent state to another consistent state. The *Isolation* property ensures that other operations cannot access or see data that have been modified during an uncommitted transaction. *Durability* defines the ability of the DBMS to recover committed transaction updates against any kind of system failure (hardware or software), i.e. once the user or application has been notified of a transaction's success the transaction will not be lost. In practice, Atomicity, Consistency and Durability are the most relevant properties for describing the dependability of a transactional system.

The ACID tests in TPC benchmarks test the system's ability to preserve the effects of committed transactions and insure database consistency after recovery from any of the following failures: **permanent irrecoverable failure of any single durable medium** (i.e., media that contain database tables and recovery redo log; either an inherently non-volatile medium, e.g., magnetic disk, magnetic tape, optical disk, etc., or a volatile medium with its own self-contained power supply); **instantaneous interruption in processing** (e.g., system crash and system hang); and **failure of all or part of main memory**, as described in Section 3. The TPC ACID tests, however, do not currently cover failures in all components of complex systems. In recent years, massive scale-out solutions configurations have become a popular way to increase performance in TPC benchmarks. As a consequence, more components are involved. Examples of hardware and software failures that can occur in such systems and that are not covered currently by the TPC ACID tests are:

- Loss or high failure rate of a storage controllers and Host Channel Adapter (HCA) attaching servers to the storage subsystem;
- Loss (or high failure rate) of a SAN switches;
- Connectivity failures, total or partial;
- Database running out of resources (e.g. memory or disk space to hold additional data);
- Application failure (e.g. an application attempts to post data that violates a rule that the database itself enforces, such as attempting to create a new account without supplying an account number);
- The Operating System (OS) running out of resources (e.g. swap space); and
- Operator error, such as terminating connections, deleting tables, etc.

The ACID tests in TPC benchmarks only guarantee the functional correctness of the DBMS' mechanisms to protect any application from failures. In fact, the TPC ACID tests do not adequately differentiate the performance impact of certain failures. In other words, except for TPC-E that partially measures the recovery time of some system failures, TPC benchmarks merely require functional correctness of such recoveries. Since all TPC benchmarks require ACID properties, what they measure is the general overhead of running a DBMS that has ACID properties activated. For instance, one of the durability tests in TPC-H is to simulate hardware failures of log and data drives. As many benchmark publications use mirrored disks to protect from such failures, they pass the durability tests without any further action. However, mirroring log and data disks has performance impacts on how fast logs and data blocks can be written. Additionally, it increases the overall cost of the system since more disks need to be configured. Alternatively, some benchmark publications do not mirror log and data disks but rely on database backups to recover from such disk failure. As a consequence the DBMS recovery time is reported as part of the load time, which is a secondary metric in TPC-H.

In this paper we argue that in this day and age systems should be tuned for a more comprehensive suite of dependability tests, and that dependability metrics should be part of TPC benchmark publications. The paper discusses WHY and HOW this can be achieved. Two approaches are introduced and discussed: 1) extending each TPC benchmark specification individually, in a customized way that takes into account the

specificities of the different TPC benchmarks; and 2) pursuing a unified approach, defining a generic specification that would be applicable to any TPC benchmark, as a complement to the already existing standard specifications. In both approaches we propose the extension of the ACID tests as a straightforward alternative for defining dependability tests required for gathering dependability related metrics.

The outline of the paper is as follows. Section 2 introduces the dependability benchmarking concept. Section 3 overviews current dependability tests in TPC benchmarks. Section 4 discusses how to extend the specification of a TPC benchmark in order to include dependability aspects, while Section 5 discusses a unified approach for augmenting TPC benchmarks as a whole. Finally, Section 6 presents the conclusion and proposes future steps.

2 The Dependability Benchmarking Concept

A dependability benchmark is a standardized specification of a procedure that merges concepts and techniques from performance benchmarking and experimental dependability assessment to evaluate and compare the dependability and performance of computer systems or computer components [12]. Dependability benchmarking is mainly inspired on experimental dependability evaluation techniques, and relies on subjecting the system to faults, while executing a typical workload, to assess the behavior of the system as failures occur.

TPC benchmarks include two major components: a workload and a set of performance metrics. To provide a practical way to measure and compare both performance and dependability of typical transactional systems, TPC benchmarks have to be enhanced with two additional elements [13]: dependability related metrics, and a faultload. Figure 1 shows the key components of current performance benchmarks and their augmented version considering dependability aspects.

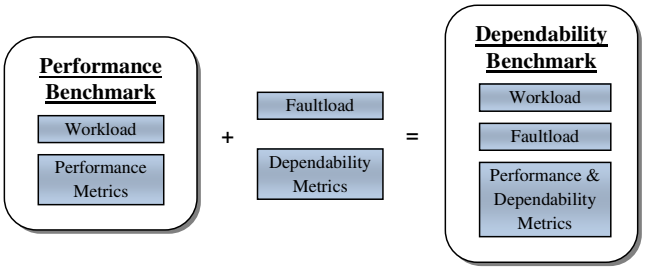

Fig. 1. Main components of a TPC benchmark enhanced to include dependability aspects

The main components of a dependability benchmark are then:

- *Metrics*: characterize the performance and dependability of the system under benchmarking in the presence of the faultload, while executing the workload.
- *Workload*: the work that the system must perform during the benchmark run, which emulates real work performed by systems in the field.
- *Faultload*: a set of faults and stressful conditions that emulate real faults experienced by systems in the field.

- *Benchmark procedure and rules*: the description of the setup, procedures and rules that must be followed during benchmark implementation/execution.

2.1 Metrics

TPC performance benchmarks rely on pure experimental approaches and provide a reduced set of metrics, making it easy for the users to obtain and understand the benchmark results, and allowing the comparison between different systems in the benchmark domain. This is also the approach for dependability benchmarking, where only direct experimental measures should be considered, following the traditional benchmarking philosophy based on a pure experimental approach.

Experimental measurements result in values that are directly related to the conditions disclosed in the benchmark report, which presents the benchmark implementation and execution details. So, similarly to performance benchmarking results (where performance values reported do not represent an absolute measure of performance, but are meant to compare computer performance in a meaningful way), the dependability benchmark results must also be understood as relative, but vitally useful to characterize system dependability in a comparative fashion (e.g., to compare alternative systems or components) or to improve/ tune system dependability. Dependability benchmark metrics for transactional environments should:

- Report the impact of the faultload on the service provided by the system, as perceived by end-users (real end-user and database administrators)
- Be easy to implement and understand by the benchmark users, and experimental values for these metrics must be simple to obtain.
- Allow an easy comparison of systems or components within the benchmark domain, in both dependability and performance aspects.
- Not be extrapolated or inferred: values must be directly computed based on the information collected during the benchmark execution.

Extending TPC benchmarks for dependability requires, in addition to the performance metrics already considered, explicit dependability related metrics, such as availability from the user's point of view or number of integrity errors detected. Values for the performance metrics can also be obtained both in the presence and absence of faults, in order to allow the evaluation of the faultload impact on the system performance.

2.2 Faultload

The faultload represents a set of faults and stressful conditions that emulates real faults experienced by the systems in the field. A faultload can be based on one or more of the following classes of faults [13]:

- *Operator faults*: operator faults in transactional systems are database administrator mistakes. The great complexity of database administration tasks and the need for tuning and administration in a daily basis clearly illustrate why these faults are prevalent in database centric systems.
- *Software faults*: software faults (i.e., program defects or bugs) are recognized as an important source of system outages, and given the huge complexity of today's software the weight of software faults tends to increase. Recent

results have shown that it is possible to emulate software faults according to the software fault classes most commonly found in field data.

- *Hardware faults*: traditional hardware faults, such as bit-flips and stuck-at, are not generally considered an important class of faults in OLTP systems, as these faults are usually related to external interferences that, normally, do not affect this class of systems. On the other hand, high-level hardware failures such as hard disk failures, network interface failures, or failures of the interconnection network itself are quite frequent. In fact, most of the transactional systems available today have recovery mechanisms explicitly developed to recover from hardware components failures.

The faultload can be based on a single class or include a mixture of faults of different classes. Note that each class comprises many types of faults. For example, the operator faults class includes a large number of types of faults, related to the memory and processes administration, the storage management, the database objects (e.g., tables, views, etc.) administration, etc. Due to the great complexity of modern systems administration, operator faults are becoming the main source of computer failures. Consequently, we believe a faultload primarily focused on this class of faults is the best approach for extending TPC benchmarks.

3 Levels of Dependability Testing in TPC Benchmarks

One of the commonalities of all current TPC benchmarks is that they require passing a set of ACID tests. Atomicity guarantees that no partially-completed operations should leave any effects on the data. Consistency is the property of the application that requires any execution of transactions to take the database from one consistent state to another. Isolation pertains to the extent to which operations done upon data by one transaction are "seen" by, or protected from, a different concurrently-running transaction or query. Atomicity and consistency are required, as the database must not be corrupted by partially completed updates, since reloading is not a viable option for repair. Isolation is needed, albeit at a reduced level, because queries may be submitted while the database is being updated and should not see uncommitted updates. Durability is essential as well, as reloading the database and replaying updates manually is not a practical means of dealing with media or system failures.

TPC-C, TPC-H and TPC-E specifications define different ACI tests but conceptually they all achieve the same. In the case of TPC-C and TPC-E ACI properties are demonstrated on the test database. Although the ACI properties are not exercised for all transaction types, the ACI properties are satisfied for all transactions. In TPC-C and TPC-E the ACI properties can be demonstrated on any one system for which results have been submitted to the TPC, provided that they use the same software executables (e.g. Operating System, database manager, transaction programs). Since TPC-H is not a transaction processing benchmark, the ACID properties are demonstrated outside the timed portion of the test using a series of transactions, not part of the benchmark workload, on a qualification database (1GB in size) irrespective of the size of the database size (scale factor). The qualification database is intended to be identical to the test database in virtually every regard, except size.

Durability tests in TPC benchmarks guarantee the ability to preserve the effects of committed transactions and ensure database consistency after recovery from any of the failures considered in the specifications. They are not intended to be exhaustive quality or dependability assurance tests, but instead aim to ensure that the System Under Test (SUT) has no unrecoverable Single Points of Failure. The current durability requirement does not take into consideration the ability of protection against the effect of multiple failures.

In TPC benchmarks there are three types of failures tested to ensure the durability of the database: Durable Medium Failure, Processing Failure, and Loss of All External Power to the SUT.

Durable Medium Failure: Durable Medium is defined as a field replaceable unit data storage medium, that is either a inherently non-volatile medium (e.g., magnetic disk, magnetic tape, optical disk, etc.) or volatile medium that will ensure the transfer of data automatically, before any data is lost, to an inherently non-volatile medium after the failure of external power, independently of reapplication of external power.

Typically, recovery from a permanent failure of durable media containing (i) database tables and (ii) log files are demonstrated. As a durable medium can fail, this is usually protected against by replication on a second durable medium (e.g., mirroring) or database logging to another durable medium. Memory can be considered a durable medium if it can preserve data long enough to satisfy the requirement, if it is accompanied by an Uninterruptible Power Supply, and the contents of memory can be transferred to an inherently non-volatile medium during the failure. Note that no distinction is made between main memory and memory performing similar permanent or temporary data storage in other parts of the system (e.g., disk controller caches).

In TPC-C, the durable media failure test(s) may be performed on a subset of the SUT configuration on a scaled down database. The tpmC for the test run must be at least 10% of the tpmC reported for the benchmark. In TPC-E, the tests must be performed on a fully scaled database at least 95% of the reported throughput with no errors, and must satisfy the transaction mix requirements. As for TPC-H, the durable media failure test(s) are preformed on a qualification database using a series of transactions that are not part of the benchmark workload, but are expressly specified for the ACID tests.

Processing Failure: Instantaneous interruption (system or subsystem crash/system hang) in processing, which causes all or part of the processing of atomic transactions to halt. This may imply abnormal system shutdown, which requires loading of a fresh copy of the operating system from the boot device. It does not necessarily imply loss of volatile memory. When the recovery mechanism relies on the pre-failure contents of volatile memory, the means used to avoid the loss of volatile memory (e.g., an Uninterruptible Power Supply) must be included in the system cost calculation. A sample mechanism to survive an instantaneous interruption in processing is an undo/redo log. In configurations where more than one instance of an operating system can participate in an atomic transaction (e.g., database cluster) and are connected via a physical medium other than an integrated bus (e.g., bus extender cable, high speed LAN, or other connection methods between the multiple instances of the operating system that could be vulnerable to a loss from physical disruption), the instantaneous interruption of this communication is included in this definition as an item that needs to be tested. Interruption of one instance of redundant connections is required.

In TPC-C, these tests have to be performed under full terminal load and a fully scaled database, and at least 90% of the tpmC reported for the benchmark. In TPC-E, the tests must be performed on a fully scaled database at, or above, 95% of the reported throughput, with no errors, and must satisfy the transaction mix requirements. In the case of TPC-H the durable media failure test(s) are preformed on a 1GB qualification database, again using a series of transactions that are expressly specified for the ACID tests.

Loss of All External Power to the SUT: When demonstrating Durability in the event of loss of all external power to the SUT, all portions of the SUT must lose power simultaneously. The power failure requirement can be satisfied by pricing sufficient UPS's to guarantee the availability of all components that fall under the power failure requirement for a period of at least 30 minutes. However, the use of a UPS protected configuration must not introduce new single points of failure (i.e., points of failure not protected by other parts of the configuration). The 30-minute requirement may be proven either through a measurement or through a calculation of the 30-minute power requirements (in watts) for the protected portion of the SUT, multiplied by 1.4.

Tests to demonstrate recovery from this failure for TPC-C, TPC-E and TPC-H are performed under same conditions as the previous section.

Table 1. Durability tests in TPC benchmarks

Benchmark Specification	Durable Medium Failure	Processing Failure	Loss of All External Power to the SUT
C	=>10% of the tpmC	=>90% of the tpmC	=>90% of the tpmC
E	=> 95% of the tpsE	=> 95% of the tpsE	=> 95% of the tpsE
H	On a qualification database (1GB)	On a qualification database (1GB)	On a qualification database (1GB)

Table 1 summarizes the Durability tests available in TPC benchmarks. In addition to the tests presented before, TPC-E requires test sponsors to report how long it takes after a catastrophic failure to get the database back up and running at 95% of the reported throughput. It defines three recovery times:

- **Database Recovery Time:** the time of recovering the database from a single failure. The start of Database Recovery is the time at which database files are first accessed by a process that has knowledge of the contents of the files and has the intent to recover the database or issue Transactions against the database. The end of Database Recovery is the time at which database files have been recovered.
- **Application Recovery Time:** the time of recovering the business application after a single failure and reaching a point where the business meets certain operational criteria. The start of Application Recovery is the time when the first Transaction is submitted after the start of Database Recovery. The end of Application Recovery is the first time, T, after the start of Application Recovery at which the following conditions are met:
    - The one-minute average tpsE (i.e., average tpsE over the interval from T to T + 1 minute) is greater than or equal to 95% of Reported Throughput.

  - The 20-minute average tpsE (i.e. average tpsE over the interval from T to T + 20 minutes) is greater than or equal to 95% of Reported Throughput.
- **Business Recovery:** the process of recovering from a single failure and reaching a point where the business meets certain operational criteria.

4 Extending the Specification of TPC Benchmarks

A potential approach towards including dependability in TPC initiatives is to extend the specification of each individual benchmark. This can be achieved by directly modifying the specification (to include additional dependability related clauses) or by defining an addendum to that specification (that specifies the additionally clauses in an independent manner). In practice, the idea is to use the basic setup, the workload, and the performance metrics specified in the TPC specification, and introduce the new components mentioned before: metrics related to dependability, and faultload. Our proposal is to extend existing ACID tests with cases that emulate operator faults, and to use metrics that reflect both performance and key dependability attributes in the presences of faults.

The key advantage of extending each TPC specification is that the dependability metrics and the faultload can be tailored to the specificities of the systems in the benchmark domain. In fact, for each benchmark, this approach allows to consider the most relevant dependability metrics and also to include the most representative faults, without concerns regarding portability across different benchmarks (and consequently, across different benchmark domains). On the other hand, this approach requires repeating the definition and approval process for each benchmark, which may be a long-term endeavor.

4.1 Including Dependability Metrics in TPC Benchmarks

We propose including dependability assessment in TPC benchmarks by considering three groups of metrics: baseline performance metrics, performance metrics in the presence of faults, and dependability related metrics. While extending the specification of a particular TPC benchmark, metrics can be defined in such a way that they take into account the specificities (e.g., service provided) of the systems in the benchmark domain (i.e., an extended TPC-C would have dependability metrics different from an extended TPC-H).

The **baseline performance metrics** are the ones that already exist in the TPC performance benchmarks. These metrics typically include the number of operations per unit of time (throughput) and the price-per-operation (cost). Note that in the context of an extended TPC benchmark, these metrics should represent a baseline performance, instead of optimized pure performance (as it is the case nowadays), and should consider a good compromise between performance and dependability.

The **performance metrics in the presence of the faultload** characterize the impact of faults on the transaction execution. Similarly to baseline performance metrics, performance metrics in the presence of the faultload should characterize throughput and cost. The first portrays the number of transactions executed per unit of time in the presence of the faultload (measures the impact of faults in the performance and favors

systems with higher capability of tolerating faults, fast recovery time, etc), and the second represents the impact of faults in the price-per-operation (measures the relative benefit of including fault handling mechanisms in target systems in terms of price).

The goal of the **dependability metrics** is to assess and evaluate specific aspects of the system dependability. An obvious problem is that dependability includes many attributes, for which tens of metrics could be defined. However, benchmarks providing a large set of measures lead to too many dimensions for results analysis, which ultimately makes the results useless. Obviously, as we are also proposing the extension of ACID tests as an alternative to include dependability attributes, the dependability metrics should focus on aspects that are observable during the execution of those tests. Examples of such metrics are recovery time, number of data integrity violations, uptime, protection against operator faults, among others.

4.2 Defining the Faultload by Extending the ACID Tests

The faultload represents a set of faults and stressful conditions that emulates real faults experienced by OLTP systems in the field. As mentioned before, a faultload can be based on three major types of faults: operator faults, software faults, and hardware faults. Although some of the published studies on the analysis of computer failures in the field are not directly focused on transactional systems, available studies clearly point operator faults as the most important cause for computer system failures [17],[18]. This way, our proposal is to augment the ACID tests by including situations that emulate operator mistakes.

Operator faults in database systems are database administrator mistakes. The great complexity of database administration tasks and the need of tuning and administration in a daily basis, clearly explains why operator faults (i.e., wrong database administrator actions) are prevalent in database systems. Obviously, end-user errors are not considered, as the end-user actions do not affect directly the dependability of DBMS. Database administrators manage all aspects of DBMS and, in spite of constant efforts to introduce self-maintaining and self-administering features in DBMS, database administration still is a job heavily based on human operators.

The emulation of operator faults in a DBMS can be straightforwardly achieved by reproducing common database administrator mistakes, i.e., operator faults can be injected in the system by using exactly the same means used in the field by the real database administrator. Different DBMS include different sets of administration tasks and consequently have different sets of possible operator faults. However, as shown in [13], it is possible to establish equivalence among many operator faults in different DBMS. In other words, a faultload based on operator faults is fairly portable across typical OLTP systems (see [13] for a detailed discussion on operator faults portability in three leading DBMS, respectively Oracle 8i, Sybase Adaptive Server 12.5, and Informix Dynamic Server 9.3. Furthermore, operator faults also emulate the high-level hardware access failures (e.g., disk access failures, network access failures, etc) normally found in OLTP environments.

The types of faults to be considered have to be chosen based on a estimation of the probability of occurrence, ability to emulate the effects of other types of faults (to improve the faultload representativeness), diversity of impact in the system, complexity of required recovery, and portability. The faultload should then be composed by a number of faults from diverse types.

4.3 The DBench-OLTP Dependability Benchmark, Example of Previous Work

The DBench-OLTP dependability benchmark, based on TPC-C, is a good example of a dependability benchmark [12]. We believe that a similar approach to include dependability evaluation can be used for other TPC benchmarks. DBench-OLTP uses the basic setup, the workload, and the performance measures specified in TPC-C and introduces the new components mentioned before: **measures related to dependability** and the **faultload**. This section presents an overview of the DBench-OLTP benchmark (adapted from [14]). A detailed description and the complete specification can be found at [13].

A DBench-OLTP dependability benchmark run includes two main phases. In Phase 1, which corresponds to a normal TPC-C execution, the TPC-C workload is run without any faults, and the goal is to collect baseline performance metrics. During Phase 2, the TPC-C workload is run in the presence of the faultload to measure the impact of faults on specific aspects of the target system dependability. As shown in Figure 1, Phase 2 is composed by several fault injection slots.

The DBench-OLTP benchmark can be used considering three different faultloads each one based on a different class of faults, namely: 1) **operator faults** (i.e., database administrator mistakes); 2) **software faults** (i.e., program bugs at the operating system level); or 3) **high-level hardware failures** (e.g., hard disk failures, power failures, etc). A general faultload that combines the three classes is also possible.

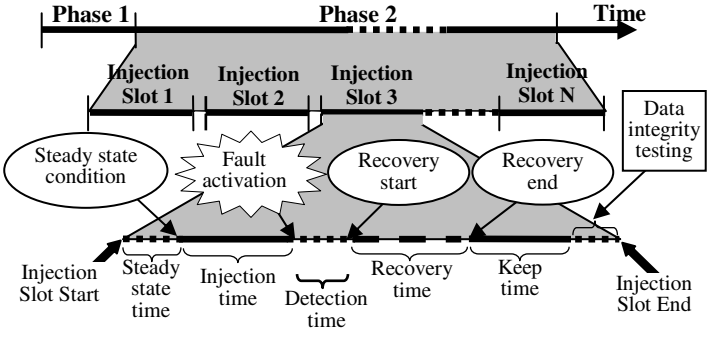

Fig. 2. Benchmark run and injection slots

The DBench-OLTP specification follows the well-accepted style of the TPC-C standard specification, and is structured in clauses that define and specify how to implement the different components of the benchmark. Briefly, the structure of the DBench-OLTP dependability benchmark specification is as follows (see [13]):

- *Clause 1. Preamble*: This clause provides an introduction to the DBench-OLTP benchmark and to the benchmark specification.
- *Clause 2. Benchmark Setup*: The benchmark setup is presented in this clause. The following elements of the setup are defined: Test configuration, System Under Test (SUT), Driver System, and Driver System/SUT Communications Interface.

- *Clause 3. Benchmark Procedure*: The benchmarking procedure, the phase 1 and phase 2 requirements, and the integrity testing requirements are presented in clause 3.
- *Clause 4. Metrics*: This clause defines the metrics provided by the benchmark and gives some guidelines on how to compute those metrics.
- *Clause 5. Faultload*: Clause 5 presents the fault types that compose faultload and provides detailed guidelines to define and implement the faultload. The steps needed to inject the faults are also presented.
- *Clause 6. Full Disclosure Report*: Clause 6 specifies what needs to be included in the full disclosure report. Like in TPC-C performance benchmark, the DBench-OLTP benchmark requires all the aspects concerning the benchmark implementation to be disclosed together with the benchmark results.

5 Unified Approach for Augmenting TPC Benchmarks

In this section, we examine another approach to include a dependability component into TPC benchmarks. As mentioned earlier, all TPC benchmarks have some dependability elements included implicitly in the specification. The previous section discussed some of the alternatives for including and measuring dependability separately in each of the TPC benchmarks. This section explores ideas on how to define a unified approach to dependability that is applicable to all TPC benchmarks.

A unified approach would mean defining a set of criteria or tests that would be carried out on the benchmark system and a methodology to measure and report the test outcomes. This approach was successfully demonstrated last year by the TPC-Energy specification [15], which defines an Energy Metric that can be measured and reported on any TPC benchmark.

Defining a specification and being able to include it uniformly in all benchmarks has several advantages. The "define-once-use-many-times" has obvious cost-savings advantages in terms of saving time in defining and implementing the specification. Additionally, if defined across all benchmarks, it is easier for the benchmark sponsor, from the execution-viewpoint, to implement it for multiple benchmarks. The specification is easier to maintain and to extend for future benchmarks. It promotes comparison across vendors and possibly even across benchmarks for those using the benchmark results to evaluate systems.

However, there are also multiple challenges in defining a global specification. For all the existing benchmarks, which have their own schemas and workloads already defined, it will be a difficult to work within the constraints to come up with tests that can be applied uniformly. Modifications to schemas and workload are not possible as that would affect the current set of TPC results, published to-date and render them non-comparable to future ones. If additions to schemas and workload are envisioned, these will have to be carried out carefully to ensure that the existing primary metrics are not impacted in any way. These constraints are likely to impose a limit on the scope of dependability metrics that could be included in a unified Dependability Specification.

Let us now look into how dependability metrics can be integrated as a metric in all TPC benchmarks. In including a measure of dependability for a benchmark, two possible methodologies are considered. The first approach is to think of dependability

as a set of features or functionality that a system possesses. If the feature does exist on the benchmark system, it is flagged in a way such that its existence is noted. The second is to define a series of tests or a workload that "measures" dependability and constitutes a secondary metric for the benchmark. While the first approach only tags the existence of dependability features, the second method not only reports their existence but also measures their performance. These two methods are examined in the remaining of this section.

5.1 Dependency Level Approach

In the Dependency Level approach dependability benchmarking is defined as a set of tests that a benchmark sponsor must execute to proof the existence of a set of dependability functionalities. We do not define a dependability metric, which could be derived from this set of tests. Rather, we define a reporting metric, called "Dependency Level". The unit of the dependency level is a number indicating how "dependent" a system is (i.e., level 1, level 2, etc.). A higher number indicates a higher level of dependability of the system. We define the following levels:

Level 1: System is "available" (i.e., not shut down and restarted) through the load of the database and performance runs in the benchmark
Level 2: Level 1 + ACID tests demonstrated on the test database;
Level 3: Level 2 + Recovery times for system hardware, operating system reboot and database recovery reported during the crash durability test;
Level 4: Level 3 + Reporting of any hardware, software or database errors during the audit runs.
Level 5: Level 4 + Protection against some user level errors (i.e., operator faults) demonstrated by the execution of additional tests during ACID.

The list of functionality defined above is generic and can be easily applied to all TPC benchmarks. No modifications or additions are required to any of the benchmark database schema. For each level, a definition of the requirements to meet all the stated functionality would have to be defined. In case of some levels such as level 5, additional tests would have to be defined in a generic manner so that they are benchmark independent. An example of a user-level error test is the dropping of a "small" database table. Small could be defined as having at most a certain percentage of the overall database size. All three TPC benchmarks have tables with various sizes so that one could define the percentage such that one table in each benchmark qualifies.

The dependency level approach has several advantages. It is easier to define and apply uniformly across all TPC benchmarks. Ease of benchmarking would be maintained as reporting the dependency level could be left to the discretion of the benchmark sponsor. Such an optional reporting has been proven advantageous by the recent introduction of optional energy reporting. However, this may also be a negative as there would be no concrete inducement for a benchmark sponsor to report at a higher dependency level.

Although most of the examples quoted above for dependency levels are easy to implement there are some that would be more difficult to define. A dependability feature like requirements for error monitoring is more variable across vendors as there are many different levels and types of error reporting on a system. Description of

what errors must be reported may be complex, especially as all vendors would be required to report at the same level of detail and granularity.

Overall, it appears that the Dependency level approach could be implemented fairly quickly and uniformly across all specifications, depending on the number of levels and dependability metrics defined in these levels. Its main deficiency is that it is limited in use, as it does not report a numerical metric that can be quantitatively compared across vendors.

5.2 Dependability Metric Approach

The second approach is to have a Dependability Metric, which would be reported as a secondary metric for all TPC benchmarks. For such a dependability metric, a set of dependability metrics have to be identified and tests would have to be created for each of these features. The tests would include the definition of the workload or fault-load that needs to be applied to the system and their elapsed times. Assuming that the faultload (see also Section 1.2) consisted of a set of tests, some sort of weighted average of the elapsed times would define the dependability metric for the benchmarks. Defining each test must include the following steps:

- Define a dependability feature that has to be measured
- Define a test that would adequately measure the feature, in the context of all existing benchmarks
- Define the measurement interval for the test

Once several such dependability features have been identified, a secondary metric that combines the measurements of all the tests will have to be defined. This could simply be a sum or an average, weighted or otherwise of all the measurements (see [16] for a detailed discussion on metric selection).

As examples, let's examine two dependability features that were also examined in the previous section: recovery time and protection against user-level errors. Recovery times for the system hardware, operating system and database are useful metrics. They indicate how quickly a system can recover from failures. These recovery times can be extracted from a simple extension of the system crash test during the ACID test (i.e. durability test), which already exists in all TPC benchmarks. It would not require any changes to the existing database specifications and would be supplementary data that would be required to be collected during the crash test. TPC-E already includes some requirements around recovery times for the database. This test can be augmented easily in a similar way to measure each of these recovery times.

Another example of a dependability measure that can be applied uniformly across benchmarks is protection against user-level errors. A test could be added to the existing ACID tests to drop a small table in the benchmark and measure the time it takes to drop and restore the table (i.e., enable the database to use the table). This would not require a change to the schema or workload. The operator fault test can be specified by using a small table in the schema of each benchmark. Again "small" would need to be defined. The choice of the table to be dropped and subsequently added back to the database can be specified in terms of its minimum size and need not be common across all benchmarks, for ease of use.

Including dependability as a secondary metric provides many advantages to the customer over specifying a dependability level. It provides a basis for numeric

comparison, which is easier to use. It would drive improvements in the dependability features since they would now be measured and hence could be compared across published results. Including and measuring the dependability features would drive benchmarking to be more representative of customer environments.

The metric-based approach is however more challenging to define, both in terms of ensuring a broad coverage of dependability measures in the metric and in defining tests that work across the entire spectrum of TPC benchmarks. Augmenting ACID tests is an obvious approach but additional tests may have to include additions to the schema. This could be complex to implement and also add to the complexity of execution of the benchmark.

The tradeoffs of difficulty of implementation and complexity of execution should be weighed against a more useful metric when deciding between Dependency Level and Dependability Metric.

6 Conclusion and Future Steps

This paper discussed different approaches for extending TPC benchmarks with dependability measures. This is a key aspect for the future of the TPC benchmark standards, as it is clear that the industry demands metrics and methodologies for measuring dependability of transactional systems. The paper discussed WHY and HOW this could be achieved for TPC benchmarks, following two different approaches: augmenting each TPC benchmark in a customized way (i.e., extending each specification individually) and pursuing a unified approach (i.e., defining a generic specification that could be applied to any TPC benchmark). Both approaches include the extension of existing ACID tests in order to allow assessing specific dependability features.

While the first approach allows defining metrics specifically targeting the systems in the benchmarking domain, as demonstrated by the DBench-OLTP benchmark, it requires revising the current specification of each TPC benchmark. On the other hand, the second approach allows a "define-once-use-many-times" that has obvious advantages in terms of saving time in defining and implementing the specification. In fact, the specification is easier to maintain and to extend for future benchmarks. The drawback is that a unified approach limits the scope of the benchmark metrics, as it must be based in the common features of the existing benchmarks.

Based on our analysis we argue that the TPC should envisage the inclusion of dependability metrics in its benchmarks following an incremental approach. Starting from a single key metric, the unified approach could be applied to broadly disseminate the concept and foster the interest of vendors and purchasers on using this type of metrics for system comparison. Afterwards, it could be extended to include more metrics. Finally, after stabilizing the concept and raising interest on dependability aspects, some TPC benchmarks could be augmented to include the most relevant dependability metrics for the systems in each particular benchmarking domain. As future work we will research the steps required to help TPC implementing these steps.

Acknowledgement

The authors would like to thank the past and present members of the TPC for their contributions in defining the methodologies to test the ACID properties, the most important concept of database management systems. Special thanks to Satinder Sethi

and Ray Glasstone for their support in writing this paper, and Greg Blotter for his comments and feedback. Last, but not least, many thanks to Prof. Henrique Madeira from the University of Coimbra, for his major contribution to the field of dependability benchmarking. His contribution and inspiring ideas contributed directly to the content presented in this paper.

References

1. Transaction Processing Performance Council, http://www.tpc.org/
2. IFIP WG10.4 on Dependable Computing And Fault Tolerance, http://www.dependability.org/wg10.4/
3. Laprie, J.C.: Dependable Computing: Concepts, Limits, Challenges. In: 25th Int. Symp. On Fault-Tolerant Computing: FTCS-25. IEEE Press, Los Alamitos
4. Avizienis, A., Laprie, J.-C., Randell, B.: Fundamental Concepts of Dependability: LAAS Research Report, N°1145 (April 2001)
5. Trivedi, K.S., Haverkort, B.R., Rindos, A., Mainkar, V.: Methods and Tools for Reliability and Performability: Problems and Perspectives. In: 7th Intl. Conf. on Techniques and Tools for Computer Performance Evaluation
6. Jenn, E., Arlat, J., Rimén, M., Ohlsson, J., Karlsson, J.: Fault Injection into VHDL Models: The MEFISTO Tool. In: Randell, B., Laprie, J.-C., Kopetz, H., Littlewood, B. (eds.) Predictably Dependable Computing Systems
7. Gray, J.: A Census of Tandem System Availability Between 1985 and 1990. IEEE Transactions on Reliability R-39(4), 409–418 (1990)
8. Hsueh, M.C., Tsai, T.K., Iyer, R.K.: Fault Injection Techniques and Tools. IEEE Computer 30(4), 75–82 (1997)
9. Carreira, J., Madeira, H., Silva, J.G.: Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers. IEEE Trans. on Software Engineering 24(2), 125–136 (1998)
10. Koopman, P., DeVale, J.: Comparing the Robustness of POSIX Operating Systems. In: 29th International Symposium on Fault-Tolerant Computing, FTCS-29, pp. 30–37
11. Arlat, J., Fabre, J.-C., Rodríguez, M., Salles, F.: Dependability of COTS Microkernel-based Systems. IEEE Transactions on Computers 51(2) (2002)
12. Vieira, M., Madeira, H.: A Dependability Benchmark for OLTP Application Environments. In: VLDB 2003 (2003)
13. Vieira, M.: Dependability Benchmarking for Transactional Systems, PhD Thesis, University of Coimbra, Portugal (2005)
14. Vieira, M., Madeira, H.: From Performance to Dependability Benchmarking: A Mandatory Path. In: Nambiar, R.O., Poess, M. (eds.) TPCTC 2009. LNCS, vol. 5895, pp. 67–83. Springer, Heidelberg (2009)
15. Poess, M., Nambiar, R.O., Vaid, K., Stephens, J.M., Huppler, K., Haines, E.: Energy Benchmarks: A Detailed Analysis. In: E-Energy 2010. ACM, New York (2010) ISBN: 978-1-4503-0042-1
16. Crolotte, A.: Issues in Benchmark Metric Selection. In: Nambiar, R.O., Poess, M. (eds.) TPCTC 2009. LNCS, vol. 5895, pp. 146–152. Springer, Heidelberg (2009)
17. Lee, I., Iyer, R.K.: Software Dependability in the Tandem GUARDIAN System. IEEE Transactions on Software Engineering 21(5) (1995)
18. Kalyanakrishnam, M., Kalbarczyk, Z., Iyer, R.: Failure Data Analysis of a LAN of Windows NT Based Computers. In: Symposium on Reliable Distributed Database Systems, SRDS, vol. 18. IEEE Press, Los Alamitos (1999)

Price and the TPC

Karl Huppler

IBM MS XQK
3605 Highway 52 North
Rochester, MN 55901 USA
huppler@us.ibm.com

Abstract. The value of a benchmark metric is directly related to how relevant it is to the consumer.

The inclusion of a price/performance metric and an availability date metric in TPC benchmarks has provided added value to consumers since the first TPC benchmark was introduced in 1989. However, over time the relative value of these metrics has diminished – both because the base price of hardware and software comprises a smaller fraction of the total cost of ownership (TCO) and because TPC pricing requirements have not kept pace with changes in the industry. This paper aims to

- Highlight the strengths provided by including price/performance and availability metrics in benchmarks.
- Identify areas where the relative value of these metrics has diminished, over time.
- Propose enhancements that could return them to provide high value to the consumer.

Many of the ideas in this paper are a result of nearly a decade of discussions with many benchmark experts. It would be difficult to identify an originator for each specific suggestion. However, it is nearly certain that this is the first comprehensive list where this collection of ideas is presented.

Keywords: Price/Performance, TCO, Total Cost of Ownership, Benchmark, TPC.

1 Introduction

When the Transaction Processing Performance Council brought out its first benchmark, TPC Benchmark A (TPC-A), in November of 1989 it introduced a novel approach that has provided value for over two decades. Not only was a strict methodology developed for implementing and measuring a performance benchmark to achieve a qualified throughput capacity for a computer system, but rules for defining a comparable price were developed and the values of price/performance and the date of availability of the total system were raised to be equal metrics along side the performance capacity metric.

These requirements provided substantial value to the consumer – by ensuring that the configurations that were measured were real configurations for which orders could

R. Nambiar and M. Poess (Eds.): TPCTC 2010, LNCS 6417, pp. 73–84, 2011.

be placed and deliveries received, and by discouraging test sponsors from overloading their system with expensive components in order to achieve the highest performance score.

For availability, there were two basic rules:

1. All hardware and software had to be orderable on the day the benchmark result was announced (This generally meant that all hardware and software was also already announced to the general public.)
2. All hardware and software had to be deliverable within six months of the day the benchmark result was announced.

For price, there were three basic rules:

1. The price had to include all hardware and software for the entire configuration, including system, storage, end-user work stations, switches, software to execute the benchmark and software to develop the benchmark.
2. In an attempt to reflect part of the cost of ownership, the price had to include 5 years of maintenance for both hardware and software.
3. The price had to be one that any customer could expect to pay

In the TPC's early years, steps were taken to strengthen the value of TPC benchmark results, including the price and availability metrics [1]. The TPC's Technical Advisory Board (TAB) was tasked with evaluating claims that benchmarks were not executed correctly. TPC Policy wording was generated to encourage fair marketing practices where TPC benchmark results were concerned. An audit process was instituted to require an independent attestation of the validity of results. Benchmark language was created to preclude the use of specialized hardware or software that was developed solely for the purpose of enhancing the benchmark result.

1.1 Eroding Strengths of the Original Metrics

Throughout the past two decades, four key trends have caused the effectiveness of these metrics to diminish.

1. Advances in technology have shifted the way that hardware and software providers interact with their customers, making some of the assumptions made in 1989 less relevant
2. Similarly, advances in technology have allowed the creation of very complex consumer applications, meaning that the relatively simple benchmark applications require memory and storage configurations that are much larger than what is installed in a typical consumer environment
3. New methods of computer solution delivery have been developed that are not included in the TPC's specification or policy language
4. In addressing these issues, the TPC has tended to relax requirements standards to accommodate an industry change rather than strengthening them.

1.2 Proposed Enhancements Included in This Paper

It is worth noting that, while the descriptions of the Price/Performance and Availability Date metrics and their historical changes are based on research and fact, the proposals

to enhance the impact of these metrics are editorial in nature. They are the result of collective experiences in the benchmarking environment, but they are essentially opinion and are presented without proof. The intent of this method is to initiate conversations on these topics within the TPC and invite public feedback, as appropriate.

2 Availability Date

We begin with this metric because it is less complex than the price/performance metric and we feel that the proposals for enhancing the metric are fairly straightforward.

As noted in the introduction, it is to the TPC's credit that this is a "metric". Benchmark results are used in marketing materials and in consumer purchase decisions. It is important for the consumer to understand whether the promised performance is available right away, in 2 months or in 6 months. The TPC requires this metric to be displayed whenever TPC benchmark results are used in public materials.

Initially, the availability requirement for TPC benchmarks was that all hardware and software must be orderable from the day the benchmark result is announced and must be available for delivery within 6 months. The declared Availability Date Metric was that date when all components of the benchmark configuration could be delivered to a customer.

Three things have changed within the industry and/or within the TPC:

First, the delivery process for some software changed. Companies no longer held orders on the books for long periods of time. Instead, they chose to only allow customers to order the software when it became ready to be shipped – often via a web download. The TPC's response was to say "OK, hardware has to be orderable from day one, but for software, you only have to describe how it can be ordered by the availability date." This response was natural, but had the effect of relaxing the requirement.

This created a disparity between the way hardware availability and software availability were treated, so the second change was to say: "OK, we'll treat everyone equally, but if your product is not orderable, you have to at least say how it can be ordered at a later date and show that the same process could be used to order a similar existing product, today." This response was fair and reasonable, and the TPC deserves credit for adding the requirement that an established order process must exist, even if a component isn't orderable until a later date. However, the net affect was to relax the prior requirement.

Finally, new development processes for both hardware and software have caused the rate of product turnover and new product delivery to accelerate. Very few consumer decisions are made based on what is available in 6 months – many are made based on what is available "now". This shift has not yet been addressed by the TPC.

2.1 Availability Date – Proposed Enhancements for the TPC

An immediate approach to strengthening the Availability Date metric lies in the hands of consumers, analysts and trade press. The approach is simply to give attention to this metric; to highlight when the products for a benchmark result are already available or will soon be available and contrast that with products that won't be available

for 5-6 months. If consumers insisted on only using benchmarks that had fully available configurations, more benchmark results would be published on available product sets.

However, the main tenant of this paper is to focus on what the TPC can do to enhance its metrics. The strength of the availability metric can be improved with two simple changes. The challenge with both is that they will remove some freedom that benchmark sponsors have enjoyed.

1. Change the availability date window to 90 days. The benchmarks of the Standard Performance Evaluation Corporation (SPEC) typically require benchmarked products to be available within 90 days, demonstrating that a 90-day rule works, even though it is "nice" to have the 185-day buffer (extended from 6 months to ensure that it would be at least six months, regardless of the start date).
2. Require that all products used in producing a result be orderable on the day the result is announced – period. The TPC's software maintenance requirements already include support for product updates, so if the level of software needed to support a result is not immediately available, all that should be needed is to be able to order the predecessor level. If it is a new version that is not yet orderable, then the upgrade price from the prior version to the new version should be included. If it is a brand new product that has never been orderable, before, then it shouldn't be included in a benchmark result until the provider has enough confidence in it to allow it to be ordered. A corollary to this is that all products used in producing a benchmark result should be described to consumers somewhere other than in the benchmark result disclosure.

3 Hardware Pricing

From a surface view, the task of generating a price for a hardware configuration appears to be very straightforward: There is a list of physical components that are required to measure the benchmark; There is a price associated with each component; Multiply the price and the quantity of each component and add them together - - Trivial if all customers expect to pay "list price", or even if all suppliers expect to offer similar discounts. However, this is hardly the case. Some suppliers list their products with fairly high profit margins and frequently offer deep discounts to their customers. Some suppliers discount very little, but price their offerings very competitively. Some suppliers tend to sell their product through resellers, where the "list price" of the supplier may be 20% higher than the list price of the reseller.

To deal with this, the TPC has a good set of rules to ensure that prices are meaningful and comparable. Many of these also apply to software and maintenance, but are listed here, first:

- The entire system under test, including storage and all servers, must be priced
- Discounts are allowed, but the basis of the discount must be described
- Both prices and discounts must be available to any customer

- Comparisons of prices must use the complete price, not component subsets, because there is no guarantee that the components are used in the same way in two different benchmark results

However, over time changes have been made to the specifications and changes have occurred in the industry that reduce the overall importance of this portion of the price. In particular:

- Where TPC-A and TPC-C once required pricing of workstations as part of the "total" system, the number of simulated users grew to be so large that the workstation price overwhelmed the price of the rest of the system, so it was eliminated from the requirement. This made the actual price more relevant, but it relaxed the "total system" requirement and caused the price per unit of work to be a smaller physical value.
- TPC-C once required pricing sufficient storage to contain 180 days of archived data, but the benchmark transactions were so efficient that this required far more storage than a typical configuration would need, so the amount of space was reduced to 60 days of archived data. Again, this served to make the price more relevant, but caused the price/performance metric to have a smaller physical value.
- Benchmark applications have become highly optimized in comparison with typical consumer applications. As a result, configurations needed to support the benchmarks, even with the adjustments listed above, have far more disk and memory than typical "customer" configurations. Discounts that can be offered on such configurations could potentially be deeper than some customers would pay for more typical configurations.
- Recognizing that a specific sales organization could deliver different pricing than a more general pricing method, such as a web tool, all that is required is to identify a pricing source that any customer has access to (1-800-BUY-ACPU, for example). This method has been casually called "TPC Price Desk" The current availability requirements dictate that such a method be allowed, since benchmarks are allowed to be published prior when configurations can be ordered through a normal sales channel.

The first two of these are really reflective of the third point. TPC benchmarks are focused on a specific set of database operations. They are touted as "full system", but they do not include the complete path of a "full application". In fact, while the database activity is fairly robust, the actual "application" layer is as slim as the benchmark implementers can make it – in order to achieve more database performance. In reality, processor configurations used to measure these "database subsystem" benchmarks would process a fraction of the data processed by the benchmark, because there is a great deal of other work that they must accomplish outside of the database subsystem. Consequently, the amount of memory and storage configured to support the database is about an order of magnitude more than is typically included in a price quotation for a consumer configuration.

3.1 Hardware Pricing – Proposed Enhancements for the TPC

If we accept that the benchmark configurations are exaggerated, and that this allows for pricing that, while be normal for these configurations, could be abnormal for more typical configurations, it follows that to the TPC should find a way to price more conservative configurations. There are three ways to accomplish this:

1. Patch the benchmarks to reduce requirements, as was done with the first two bullets, above. This provides a temporary improvement, but it is cosmetic, at best.
2. Change the benchmark to become more robust and require more processing resources relative to the other resources in the configuration. This is an admirable approach, and one that partially succeeded with the introduction of TPC-E, but benchmark development takes a long time and this would diminish the data base focus that is a mainstay of TPC benchmarks.
3. Define a subset configuration to be priced, based on the measured configuration. Each benchmark would have its own rules. For example, TPC-C might require pricing a database server with 1/8 the memory used in the benchmark and 1/8 the total number of disks.

This last solution is radical, in that it diverts from the standard rule of "price what you measure, measure what you price." However, it resolves the challenges listed here quite nicely: Configurations would be of a size that customers might actually consider buying. Published benchmark prices would be self-verifying, because customers would go to their price source of choice and demand a similar or better price.. Furthermore, the methodology associated with this can be adjusted in a new version of the benchmark to reflect current buying practices and technology.

The fundamental point is to price configurations that have meaning to the consumer, thereby increasing the relevance of the price/performance metric.

4 Software Pricing

Software presents a particular challenge in pricing requirements, in that customers do not buy software. They pay a license fee for the privilege to use the software. There are often a number of options available to consumers:

- license a specific number of users to exercise the software, where a small number of sessions are expected to use the software
- license a specific number of processors or processor-cores on which the software will be exercised, where the use of the software is unlimited, as long as it is run on no more physical resources than licensed
- purchase a perpetual license, where the licensee is allowed to use the software "forever", even after maintenance support for the software is dropped
- purchase a term-limited license, where the licensee is not allowed to use the software once the term expires, unless the license is extended
- purchase of a utility-based license, where charges are based on the frequency with which the code is exercised.

Any of these options may be viable for a particular consumer scenario. However, a performance benchmark introduces the challenge of producing comparable results.

Utility-based licenses would be difficult to enforce in a benchmark environment. They are typically oriented to a very large number of potential users, each only accessing the computer for a brief period. This is in contrast to benchmark environments when the computer is running at maximum capacity.

Examining user-based licensing, the benchmark application tends to be simpler and more stream-lined than a typical consumer application, the number of active sessions needed to drive a system to full capacity may be far fewer than the number that may access similar software in a consumer environment. This is compensated in TPC-E, where the number of users licensed must be equal to the tpsE throughput value and in TPC-H, where the number of licensed users must be 10 times the number of query streams executed – with the result that almost all benchmark results use licenses that are priced by number of processors or cores.

Price by cores also has challenges. Seldom will a large system be used exclusively for database activity, and seldom will it be executing at 100% of system capacity, Consequently, there are more processing cores active during the benchmark than in almost any consumer scenario. Furthermore, all processor cores are not created equally, so many software vendors establish rules for each of several architectures that either charge less per core for some architectures than others, or that create a smaller number of "chargeable cores" than what is in the configuration. However, this is also how the licenses are presented to actual consumers, so apart from the potential for a company to provide lower prices for software running on "preferred" hardware than on other hardware, it may be as close to reality as possible.

Finally, there is the difference between perpetual and term-limited licensing. Term-limits are more similar to a lease of a license than a purchase. Note that, in the hardware discussion, the term "lease" did not surface - - This is because the TPC has successfully barred hardware leases from being used in benchmark pricing. This does not mean that a hardware lease is not a viable option for a consumer – only that it was deemed not to be comparable with hardware purchases. Even after a hardware product has been fully depreciated, the consumer has the option of retaining and exercising the hardware – This is not the case with term-limited licenses.

4.1 Software Pricing – Proposed Enhancements for the TPC

The pricing of software could be made to be much more comparable with the implementation of three enhancements:

1. Treat hardware and software consitently by requiring a "perpetual" purchase of the entire configuration, retaining some residual value for as long as the consumer desires.
2. Rather than try to compensate for the simulated number of users in a configuration, the TPC should simply disallow per-user-based or usage-based licenses and require perpetual software licenses on a per-core or per-processor basis.
3. Apply a "reality adjustment" that is similar to the proposal for storage and memory in the hardware examples. Each benchmark committee should determine the ratio of number of cores to be licensed relative to the number measured, perhaps reducing this be a factor of 2-3.

As with hardware, the latter suggestion strays from the concept of "Price what you measure; measure what you price." It is proposed in the spirit of delivering priced configurations that are consumer-based, rather than benchmark-based, to encourage pricing of configurations that are relevant to consumers.

5 Maintenance Pricing

Maintenance of hardware and software is a critical part of data center operations, and a significant component in the total cost of ownership for a computing solution. Maintenance, or the broader category of "service," is an area that is uniquely customizable to fit the consumer's needs. Almost all customers require warranty-level maintenance on their products to fix items that fail or have defects, but most customers also require a level of additional support that helps them to make optimal use of their investment. This includes such things as software upgrades, consultation, education offerings, predictive analysis and myriad other options. Because customers' needs differ, product suppliers tend to offer a variety of options.

When the TPC first defined rules for pricing, many systems were used for well over five years before being replaced with newer technology. Five years was also the fastest that the United States Internal Revenue Service and United States Generally Accepted Accounting Principles (GAAP) allowed for depreciation of assets. Similar rules were established with the International Financial Reporting Standards (IFRS) and the International Accounting Standards (IAS) This prompted the TPC to require maintenance support for a five-year period from the initial publish date. Over time, it appeared that the rapid turn-over of technology would prompt depreciation schedules to be shortened to three years, and the TPC adjusted to use that schedule.

Hardware maintenance requirements for TPC benchmarks are fairly straightforward and, in the opinion of this author, fairly close to consumer requirements. In summary, there are two methods of maintenance allowed for hardware by the benchmark specifications:

1. 7X24 support with 4-hour response time to begin working a problem, continuing work until the problem is resolved
2. For customer-replaceable parts, the option to price an on-location supply of 10% extra parts, with a mail-in replacement offering for failed parts.

Although other hardware maintenance offerings are provided by many vendors, these requirements are representative of what a consumer might choose in an environment that supports critical run-your-business applications. Not all vendors offer the second option, but virtually all vendors that sell into mission-critical environments offer the first of these.

It is more difficult to define specific levels of software support, because software also has "soft" problems – namely user errors, software configuration questions, implementation and optimization concerns, and the like. Furthermore, many software providers handle these support items in a different way. To provide a set of requirements that are common among most software vendors, the TPC chose a "lowest common denominator" approach, requiring only the most basic maintenance for software. The software maintenance requirements can be summarized as:

1. 7X24 support with a 4-hour response time to acknowledge that a software bug exists and begin working on it - - - but with no specific commitment to resolve the bug in any given amount of time
2. Access to maintenance update software, such as Service Packs, that include rolled-up groups of fixes to previously resolved software bugs.

Areas that are not included and are important to almost all customers:

- Functional software updates (new releases)
- Operational guidance
- Customer Education

The result is that often the maintenance offerings that are priced in a benchmark satisfy only the bare minimum requirements of the benchmark and do not reflect the needs of a typical consumer.

5.1 Maintenance Pricing – Proposed Enhancements for the TPC

Three enhancements would bring TPC requirements for maintenance and support to a point where it would reflect relevant information for the consumer:

1. It is first worth noting that depreciation schedules have remained at five years, for the most part. A reflection of this can be found in the United States IRS form 4562, used to document depreciation for tax purposes in the United States [2].Consequently, it is recommended that maintenance prices be extended to the TPC's original five-year requirement.
2. However, we also note that the initial purchase of the equipment is a capital investment, while the support costs are considered to be expenses – most often spread on an annual, quarterly, or monthly basis. Although the total price/performance metric should reflect the full five-year costs of initial purchase and maintenance, there should be secondary metrics that call out the initial purchase price and the on-going yearly maintenance after the first year. It should be valid to compare benchmark results based on initial purchase price, annual maintenance and total 5-year price.
3. Most important and most difficult to define: The requirements for software support should be raised to a level that is relevant to a consumer whose business depends on the products installed on the system(s). Support requirements should include the current TPC requirements, plus ongoing product updates, operational guidance and some level of customer education. Each component of maintenance should be priced for no more than one year, multiplying that value by the number of years needed (assuming some coverage is included in the purchase price for the first year) to create a five-year maintenance cost. The challenge will be drawing a line between "operational guidance" and "consultation services". Most companies offer both, but do not always use the same criteria in shifting from the moderately priced, almost always purchased option and the more expensive level of support that would include design guidance, optimization and other services that might more typically be fee-for-service options.

6 The Relationship between the "TPC Price" and TCO

The TPC does not claim that the required price sheet for their benchmarks represents the total cost of ownership of the system, although the inclusion of maintenance requirements have prompted some to call it the "cost of ownership" without the word, "total." The following table represents many of the elements of the total cost of ownership for a computer system, along with comments on whether the cost is covered or partially covered in the existing TPC requirements or in the proposed enhanced requirements.

Element	Included in TPC-Price?	Included in Recommendations?	Comment
Initial HW cost	Yes	Yes - Improved	Improved in recommendations by requiring a more "consumer-appropriate" configuration
Initial OS, Middleware, Database SW cost	Yes	Yes - Improved	Improved in recommendations by requiring perpetual licenses on a "typical" number of processor cores
Initial Application SW cost	No	No	Often a major component of the cost, but not possible to include in a generic benchmark
HW upgrades cost	No	No	Ability to upgrade without a total replacement can be a key factor in a purchase decision, but is not included, here.
OS/Middleware/Database SW upgrades cost	No	Yes	Proposed support costs include software upgrades
Application upgrades cost	No	No	Another missing element that cannot be in a generic benchmark
HW maintenance	Yes	Yes - Improved	Improved in recommendation by extending for 5 years and providing differentiation between capital and expense costs
OS/Middleware/ Database maintenance	Yes	Yes – Much Improved	Improved in same ways as hardware, plus requiring a more robust level of support that is appropriate for most consumers with mission-critical computing environments.
Application maintenance	No	No	See other "application" entries
Application set-up, customization	No	No	See other "application" entries
DB administration	No	Minor	By including an operational level of support, a small amount of DB administration costs are also included
Systems operations	No	Minor	By including an operational level of support, a small amount of Systems Operations costs are also included
Electricity	No	No, but	This growing area of cost of ownership is covered by the TPC's new TPC-Energy metrics that can be published with each benchmark at the sponsor's option. Unlike price,

			which includes variations based on channels, discounts and other marketing-related areas, the energy component of the total cost of ownership can be quantified with verifiable measurement. [3]
Floor-space, building costs	No	No	This is quantifiable area that could be a candidate for inclusion in a future price metric.
Training	No	Yes	The recommendation for support costs includes that some level of consumer education be required in the ongoing costs.
Unscheduled down-time costs	No	No	While it is difficult to quantify, this is not only a major part of the TCO, but can be extremely disruptive to the business. A proposal in the TPC-TC '09 recommended that the TPC undertake the creation of a "dependability" benchmark. [4]
Scheduled down-time costs	No	No	If a resiliency benchmark is created, it should cover both planned and unplanned down-time

7 Summary

A benchmark metric is only as valuable as it is relevant. The price/performance and availability metrics from TPC benchmarks have been quite relevant in the past, but changes in technology, changes in other elements in total cost of ownership and changes in the benchmarks themselves have reduced the worth of these metrics.

By implementing the suggestions raised in this paper, the TPC could dramatically improve the relevance of the price/performance and availability metrics of TPC benchmarks. These suggestions should be viewed as an entire set, rather than specific individual decisions. Some of these changes would affect all vendors. Others would affect only some. With the entire collection, the relative impact to each vendor is mitigated by both the affect on other vendors and the closer approximation of consumer reality.

The proposed changes focus on

- Physical configurations that are more relevant to consumers
- Availability requirements that are closer to consumer buying practices
- Software licensing requirements that represent a typical mission-critical environment, where longevity of an application is expected
- Maintenance requirements that represent the typical levels of support that a consumer would require for a run-your-business application

These changes do not approach the total cost of ownership for a computer system, but they enhance the portions of TCO that are covered by the TPC Price Specification to more realistically represent the actual contribution to TCO for these components.

Implementing these changes will take courage and clearly generate benchmark results that are not comparable with those published today. Not implementing these

changes destines the price/performance metric, and to a lesser degree the availability metric, to reside in a realm where marketing claims are made, but the actual results are ignored.

7.1 Another Alternative

Without changes such as those suggested here, the TPC could be better off without official price/performance metrics at all. It may be that the best solution is to require that a well-defined bill-of-materials be listed, and that the list price for any products that are not publicly listed through another source be included, but to leave the computation and comparisons of price to the marketing teams, rather than the performance engineers. The advantages of this are that the comparison method would allow "for purchase" comparisons, "for lease" comparisons, "utility/cloud" comparisons, "purchase only" comparisons, "total cost of ownership" comparisons, and so on. The disadvantages of this are that the methods for comparison would be unregulated and could easily create confusing or misleading information.

References

1. Shanley, K.: History and Overview of the TPC (February 1998), http://www.tpc.org/information/about/history.asp
2. United States Internal Revenue Service, Instructions for Form 4562 (2009), http://www.irs.gov/pub/irs-pdf/i4562.pdf
3. Transaction Processing Performance Council, TPC-Energy Specification (2010), http://www.tpc.org/tpc_energy/default.asp2009
4. Vielra, M., Madiera, H.: From Performance to Dependability Benchmarking: A Mandatory Path. Springer, Heidelberg, ISDN 0302-9743 (Print) 1611-3349 (Online)

Impact of Recent Hardware and Software Trends on High Performance Transaction Processing and Analytics

C. Mohan

IBM Almaden Research Center, K01/B1, 650 Harry Road, San Jose, CA 95120, USA
mohan@almaden.ibm.com

Abstract. In this paper, I survey briefly some of the recent and emerging trends in hardware and software features which impact high performance transaction processing and data analytics applications. These features include multicore processor chips, ultra large main memories, flash storage, storage class memories, database appliances, field programmable gate arrays, transactional memory, key-value stores, and cloud computing. While some applications, e.g., Web 2.0 ones, were initially built without traditional transaction processing functionality in mind, slowly system architects and designers are beginning to address such previously ignored issues. The availability, analytics and response time requirements of these applications were initially given more importance than ACID transaction semantics and resource consumption characteristics. A project at IBM Almaden is studying the implications of phase change memory on transaction processing, in the context of a key-value store. Bitemporal data management has also become an important requirement, especially for financial applications. Power consumption and heat dissipation properties are also major considerations in the emergence of modern software and hardware architectural features. Considerations relating to ease of configuration, installation, maintenance and monitoring, and improvement of total cost of ownership have resulted in database appliances becoming very popular. The MapReduce paradigm is now quite popular for large scale data analysis, in spite of the major inefficiencies associated with it.

Keywords: Analytics, Appliances, Cloud Computing, Databases, FPGAs, Hardware, Key-Value Stores, Multicore, Performance, Software, Storage Class Memories, Transaction Processing.

1 Introduction

In the last few years, there have been many important developments in the hardware and software arenas that are having serious implications on how transaction processing and data analytics are accomplished. They relate to memory and storage devices, computer architectures, programming paradigms, and software architecture, design and implementation. While some of these developments have already impacted mainstream (traditional) applications, others have been influencing more heavily relatively new types of applications (e.g., Web 2.0 ones like social networking and cloud computing). Some older technologies like field programmable gate arrays (FPGAs) have

R. Nambiar and M. Poess (Eds.): TPCTC 2010, LNCS 6417, pp. 85–92, 2011.

been adopted for building database appliances [19, 20]. In this paper, I survey briefly some of these recent and emerging trends in hardware and software architectural features which impact high performance transaction processing (HPTP) and data analytics applications.

Research groups in a number of institutions across the world are focusing on the topics covered in this paper. Representative samples are: EPFL Lausanne [28], ETH Zurich [32], Google Research [49], IBM Almaden Research Center [31], Microsoft Research [45], MIT [29], Stanford University [34], University of California at Santa Barbara [33], Yahoo! Research [37] and Yale University [30]. In addition to big companies like IBM and Oracle, a number of startups are also developing related products. Some of the startups are Netezza, ParAccel and Schooner Information Technology.

2 Multicore

While symmetric multiprocessors (SMPs) have been around for decades in the mainframe context [26], it is only relatively recently that similar concepts have begun to make an impact in the non-mainframe computer architectures. More and more chip designers are using the ability to include a significantly larger number of transistors on a chip to increase the number of processing elements, *cores*, on a single chip, thereby creating multicore systems [25, 47]. IBM's Power 795 [50] can have up to 256 cores, while Oracle's Exadata Database Machine X2-8 has 128 cores for database processing and 168 cores for storage processing [51].

It is fairly straightforward to leverage multiple cores for dealing with analytics applications by parallelizing query processing, as in traditional parallel database management systems (DBMSs) [27]. While leveraging a large number of cores would appear to be an easy objective to accomplish in a HPTP system for gaining greater scalability by running larger number of transactions in parallel, significant problems could arise, for example, from the need to maintain coherence across the much larger number of processor caches [6]. Shared data structures and how their concurrency control is handled (e.g., via latches [13]) would surely have to be rearchitected to alleviate such problems. A benchmark of four popular open-source storage managers (Shore, BerkeleyDB, MySQL and PostgreSQL) on a modern multicore machine (8 cores with 4 hardware thread contexts per core) has revealed that they all suffer in terms of scalability [4]. Work done on FPGAs to improve query processing performance has been shown to apply to multicore environments also [20]. The MapReduce paradigm [36] has become quite popular for large scale data analysis using a vast array of cheap computational engines. Google, which has been leveraging multicore systems extensively for many years, has issued a timely warning recently about the relative benefits of *brawny cores* over *wimpy cores* [48].

3 Large Main Memories

Main memory sizes have been growing by leaps and bounds. In high end machines like the IBM P7 [21, 50], it could be as high as 8 terabytes (TB), and in Oracle's Exadata Database Machine X2-8 it is 2 TB [51]. While in-memory database technology has

been around for more than 3 decades, starting with the main storage database (MSDB) feature of IMS/Fastpath to more modern relational database management systems (RDBMSs) like IBM SolidDB [16] and Oracle TimesTen [17], proper exploitation of TB size memories requires major rearchitecting of HPTP systems. Not all of the memory might be accessible at the same speed since the overall memory architecture might be NUMA like. Localizing accesses to be within a cluster of processors' memory might be beneficial from a performance viewpoint.

Not all of the existing features in a DBMS would scale well when the sizes of memories become really large. For example, in the case of Java Virtual Machines (JVMs), they are unlikely to work well beyond a few GBs of heap size due to the way most garbage collection algorithms in real life products work (this is the case in spite of all the research work that has gone on with respect to concurrent garbage collection algorithms).

If users are tempted to make their objects very large because memory sizes have become large, numerous data structures in a DBMS might not be well designed for such sizes. The analogy here is similar to what RDBMSs had to do to deal with binary large objects (BLOBs) - separate storage areas for them compared to normal fields, different recovery and locking approaches, etc [22].

From a concurrency perspective also, including huge amount of memory in a single heap would be a problem. Storage management within a heap also would need to be done differently, possibly trading off some fragmentation for speedier and less disruptive handling of storage allocations and deallocations. Key value store (KVS) systems like Memcached have already adopted different approaches to storage management because of such reasons [23, 24].

4 Storage Class Memories

For some years now, flash memory [43] has been very popular in the context of consumer devices like digital cameras and personal music players. Of late, flash is being adapted for usage in the enterprise context where some of the negative aspects of flash like data retention and other reliability properties, and asymmetry between read and write speeds have to be handled differently. Wear leveling and other techniques have been developed to deal with them. As denser chip technologies come into existence and flash encounters difficulties in working correctly under those conditions, other ways of constructing persistent memories are becoming popular. They are also intended to move away from the block oriented nature of flash, which has resulted in flash being used primarily as a disk replacement, and to let byte oriented read and write be supported which would make them behave more like DRAM. One such technology is phase change memory (PCM) [2, 3]. Hardware companies like Numonyx/Micron and Samsung are in the process of releasing PCM chips. Micron has said in July 2010 that they are in production with two 128Mb, 90nm PCM parts (Omneo P5Q and P8P) and in development with a 1Gb, 45nm PCM die.

Meanwhile, computer architects, and designers of operating systems (OSs), storage software and middleware are investigating different ways of exposing and leveraging the functionality of such chips. In IBM Almaden Research Center, we have an

exploratory research project which is focused on the middleware implications of PCM [1]. The design of a new file system for data resident in PCM is presented in [14].

5 Database Appliances

Considerations relating to ease of configuration, installation, maintenance and monitoring, and improvement of total cost of ownership have resulted in database appliances becoming very popular. Many big and small companies are producing appliance products.

In the last few years, FPGAs have become popular in building database machines/appliances by companies like Netezza and Kickfire. This has increased the level of interest in the use of FPGAs in a database context. Designers hope to get better performance and lower energy consumption through the use of FPGAs, compared to the use of traditional CPUs. FPGAs provide very high I/O bandwidth, have low power consumption and support massive parallelism.

One of the major emerging trends is the concept of workload optimized systems. IBM is seriously behind this architectural push to codesign hardware and software to optimize the overall system performance for a certain class of applications (e.g., IBM pureScale Application System for transactional workloads and IBM Smart Analytics System for data warehousing and analytics [40]).

6 Transactional Memory

Transactional memory (TM) systems have been developed to make the exploitation of the parallelism possibilities of multicore systems to be easier to accomplish by ordinary programmers [7, 8, 9]. Of course, for decades, designers of traditional TP systems (e.g., IMS/DC and CICS) and DBMSs (e.g., IMS, DB2, Oracle) have been leveraging the parallel execution capabilities of traditional multiprocessor computers to run numerous transactions and queries in parallel. Locking and latching mechanism have been carefully used to enable such executions to happen in an orderly and consistent fashion [13]. Some researchers have recently started wondering if somehow the TM features of modern hardware and software could be exploited by DBMSs to improve their performance and to ease the job of DBMS designers in correctly leveraging greater levels of parallelism [11]. Having spent 3 decades in the concurrency and recovery areas, I am very doubtful about that!

7 Key Value Stores

Web 2.0 companies have been the breeding grounds for a number of non-SQL data management systems which go by the name of key value stores (KVSs). Some of the more popular KVSs are Cassandra [38], Memcached [39], Redis [52] and Voldemort [53]. Compared to RDBMS, KVSs have limited functionality along many dimensions. They were intended to counter the high overheads of RDBMS for certain types of applications. Their APIs have lot less functionality than SQL and they typically do not support the traditional notion of transactions. Many Web 2.0 applications (e.g.,

Facebook, Twitter, Wikipedia and YouTube) have adopted KVSs as their data management platforms, especially in the context of ultra large main memory systems in order to meet stringent response time requirements. Typically, these systems do not perform logging and recovery as in traditional DBMSs [13]. They rely on in-memory replication to provide some level of fault tolerance. While some of the Web 2.0 applications were initially built without traditional transaction processing functionality in mind, slowly system architects and designers are beginning to address such previously ignored issues. The availability, analytics and response time requirements of these applications were initially given more importance than ACID transaction semantics and resource consumption characteristics.

In our project at IBM Almaden [18], we are adding transaction support to Memcached. Our project's goal is to support ultra large in-memory databases in DRAM and PCM, with high availability and high performance. Bitemporal data management [41] has also become an important requirement, especially for financial applications. The newest releases of DB2 [42] on the different platforms include support for it.

8 Cloud Computing

The promise of cloud computing of elasticity, the "pay as you go" payment model, comes with a number of additional functionality to be worried about by the providers of cloud-based services. In particular, for HPTP systems, multi-tenancy, security, quality of service guarantees via service level agreements, availability [46], and additional control and flexibility of migrating work between private and public clouds are major design issues. To provide more focus to the cloud computing area and also to provide a forum for discussions amongst researchers with different backgrounds, a new research conference, the ACM Symposium on Cloud Computing, has also been started this year (2010). In the context of the cloud environment, new benchmarks are also being defined, e.g., the Yahoo! Cloud Serving Benchmark YCSB [35].

9 Conclusions

In this paper, I summarized some of the recent and emerging trends with respect to hardware and software features that relate to high performance transaction processing and data analytics applications. This is an exciting time with significantly large number of on-going activities that span the hardware-software boundary. Consolidation in the industry, with recent acquisitions by companies like Cisco, HP, IBM and Oracle, is also contributing to this trend. More and more researchers and practitioners are being forced to get out of their traditional way of working in technical silos. Led by the internet companies, exploitation of large clusters of servers, *server farms*, is also becoming common, both in the cloud and the non-cloud environments [46]. Interesting tradeoffs involving availability, data consistency, power consumption, throughput, response time, dynamic adaptation to workload fluctuations and ease of rapid application development are being made all the time, and, as to be expected, differently in different application environments.

Readers interested in more information on the topics summarized here could follow up by making use of the numerous citations given in this paper to research papers, presentations and web sites. I have also provided some information on the various research groups and commercial organizations that are working on related technologies.

References

1. Fang, R., Hsiao, H., He, B., Mohan, C., Wang, Y.: High Performance Database Logging using Storage Class Memory (2010) (submitted for publication)
2. Freitas, R., Wilcke, W.: Storage-Class Memory: The Next Storage System Technology. IBM Journal of Research and Development 52(4), 439–447 (2008)
3. Lee, B., Ipek, E., Mutlu, O., Burger, D.: Phase Change Memory Architecture and the Quest for Scalability. Communications of the ACM 51(7) (July 2010)
4. Johnson, R., Pandis, I., Hardavellas, N., Ailamaki, A., Falsafi, B.: Shore-MT: A Scalable Storage Manager for the Multicore Era. In: Proc. 12th International Conference on Extending Data Base Technology (EDBT), St. Petersburg, Russia (March 2009)
5. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W.: Dynamo: Amazon's Highly Available Key-Value Store. In: Proc. 21st ACM Symposium on Operating Systems Principles (SOSP), Stevenson, USA (October 2007)
6. Lee, R., Ding, X., Chen, F., Lu, Q., Zhang, X.: MCC-DB: Minimizing Cache Conflicts in Muli-Core Processors for Databases. In: Proc. 35th International Conference on Very Large Data Bases (VLDB), Lyon, France (August 2009)
7. Shriraman, A., Dwarkadas, S., Scott, M.: Tapping into Parallelism with Transactional Memory. The USENIX Magazine 34(2), 12–23 (2009)
8. Porter, D., Hofmann, O., Witchel, E.: Is the Optimism in Optimistic Concurrency Warranted? In: Proc. 11th Workshop on Hot Topics in Operating Systems (HotOS), San Diego, USA (May 2007)
9. Porter, D., Witchel, E.: Understanding Transactional Memory Performance. In: Proc. IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), White Plains, USA (March 2010)
10. Pankratius, V., Adl-Tabatabai, A.-R., Otto, F.: Does Transactional Memory Keep Its Promises? Results from an Empirical Study, Technical Report 2009-12, IPD, University of Karlsruhe, Karlsruhe, Germany (September 2009)
11. Dias, R., Lourenco, J.: Unifying Memory and Database Transactions. In: Sips, H., Epema, D., Lin, H.-X. (eds.) Euro-Par 2009 Parallel Processing. LNCS, vol. 5704, pp. 349–360. Springer, Heidelberg (2009)
12. Rossbach, C., Hofmann, O., Witchel, E.: Is Transactional Programming Actually Easier? In: Proc. 15th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming (PPoPP), Bangalore, India (January 2010)
13. Mohan, C., Haderle, D., Lindsay, B., Pirahesh, H., Schwarz, P.: ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging. ACM Transactions on Database Systems 17(1), 94–162 (1992)
14. Condit, J., Nightingale, E., Frost, C., Ipek, E., Lee, B., Burger, D., Coetzee, D.: Better I/O Through Byte-Addressable, Persistent Memory. In: Proc. 23rd ACM Symposium on Operating Systems Principles (SOSP), Big Sky, USA (October 2009)

15. Ousterhout, J., Agrawal, P., Erickson, D., Kozyrakis, C., Leverich, J., Mazières, D., Mitra, S., Narayanan, A., Parulkar, G., Rosenblum, M., Rumble, S., Stratmann, E., Stutsman, R.: The case for RAMClouds: Scalable High-Performance Storage Entirely in DRAM. ACM SIGOPS Operating Systems Review 43(4), 92–105 (2010)
16. IBM SolidDB In-memory Database, http://www.ibm.com/software/data/soliddb/
17. Oracle TimesTen In-memory Database, http://www.oracle.com/database/timesten.html
18. Mohan, C.: Implications of Storage Class Memories (SCMs) on Software Architectures. Presentation at 2nd TPC Technology Conference on Performance Evaluation & Benchmarking (TPC TC), Collocated with 36th International Conference on Very Large Data Bases (VLDB), Singapore (September 2010), http://bit.ly/TPCpcm
19. White Papers, Netezza, http://www.netezza.com/products/whitepapers.aspx
20. Mueller, R., Teubner, J.: FPGA: What's in it for a Database? Tutorial. In: Proc. ACM SIGMOD International Conference on Management of Data, Providence, USA (June 2009)
21. Anselmi, G., Blanchard, B., Cho, Y., Hales, C., Quezada, M.: IBM Power 770 and 780 Technical Overview and Introduction, IBM Redpaper REDP-4639-00 (March 2010), http://www.redbooks.ibm.com/redpapers/pdfs/redp4639.pdf
22. Sears, R., van Ingen, C., Gray, J.: To BLOB or Not To BLOB: Large Object Storage in a Database or a Filesystem? Microsoft Research Technical Report MSR-TR-2006-45 (April 2006)
23. Mohan, C., Haderle, D.: Algorithms for Flexible Space Management in Transaction Systems Supporting Fine-Granularity Locking. In: Proc. 4th International Conference on Extending Database Technology, Cambridge, UK (March 1994); A longer version of this paper is available as IBM Research Report RJ9732, IBM Almaden Research Center, San Jose, USA (March 1994)
24. Yen, S.: Memcached Slab Allocator (June 2009), http://code.google.com/p/memcached/wiki/MemcachedSlabAllocator
25. Wikipedia: Multi-core Processor, http://en.wikipedia.org/wiki/Multi-core_processor
26. Wikipedia: Symmetric Multiprocessing, http://en.wikipedia.org/wiki/Symmetric_multiprocessing
27. Dewitt, D., Gray, J.: Parallel Database Systems: The Future of High Performance Database Systems. Communications of the ACM 35(6), 85–98 (1992)
28. DIAS: Data-Intensive Applications and Systems Laboratory, EPFL Lausanne, Switzerland, http://dias.epfl.ch/
29. Database Group. MIT, Cambridge, http://db.csail.mit.edu/
30. Database Research at Yale. Yale University, New Haven, http://db.cs.yale.edu/
31. Information Management, IBM Almaden Research Center, San Jose, USA, http://almaden.ibm.com/cs/disciplines/dm/
32. The Database Research Group, ETH Zurich, Switzerland, http://www.dbs.ethz.ch/
33. Distributed Systems Lab. University of California at Santa Barbara, Santa Barbara, USA, http://www.cs.ucsb.edu/~dsl/?q=projects/active
34. RAMCloud Project. Stanford University, Stanford, USA, http://fiz.stanford.edu:8081/display/ramcloud/Home

35. Cooper, B.F., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R.: Benchmarking Cloud Serving Systems with YCSB. In: ACM Symposium on Cloud Computing (SOCC), Indianapolis, USA (June 2010)
36. Wikipedia: MapReduce, http://en.wikipedia.org/wiki/MapReduce
37. Web Information Management, Yahoo! Research, http://research.yahoo.com/project/212
38. Apache Cassandra Project, http://cassandra.apache.org/
39. Wikipedia: Memcached, http://en.wikipedia.org/wiki/Memcached
40. Boike, N.: Leading Edge - IBM's Workload-Optimized Approach Puts Your Business in Front of the Competition. IBM Systems Magazine (October 2010), http://bit.ly/WOsys
41. Wikipedia: Temporal Database, http://en.wikipedia.org/wiki/Temporal_database
42. Beulke, D., and Associates: IBM DB2 10 for z/OS Beta, http://bit.ly/DB2z10
43. Wikipedia: Flash Memory, http://en.wikipedia.org/wiki/Flash_memory
44. Atwood, G.: The Evolution of Phase Change Memory, Micron White Paper (July 2010), http://www.micron.com/document_download/?documentId=5539
45. Cloud Computing Futures, Microsoft Research, Redmond, USA, http://research.microsoft.com/en-us/groups/ccf/
46. Ford, D., Labelle, F., Popovici, F., Stokely, M., Truong, V.-A., Barroso, L., Grimes, C., Quinlan, S.: Availability in Globally Distributed Storage Systems. In: Proc. 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI), Vancouver, Canada (October 2010)
47. Portal for Multicore Processor News and Information, http://www.multicoreinfo.com/
48. Hoelzle, U.: Brawny Cores Still Beat Wimpy Cores, Most of the Time. To Appear in IEEE MICRO (2010)
49. About Google Research, http://research.google.com/about.html
50. Chen, A., Cruickshank, J., Costantini, C., Haug, V., Maciel, C., Schmidt, J.: IBM Power 795 Technical Overview and Introduction - A Draft IBM Redpaper Publication, REDP-4640-00 (September 2010), http://www.redbooks.ibm.com/redpieces/pdfs/redp4640.pdf
51. Data Sheet: Oracle Exadata Database Machine X2-8, http://www.oracle.com/technetwork/database/exadata/dbmachine-x2-8-datasheet-173705.pdf
52. Redis – A Persistent Key-Value Database with Built-in Net Interface Written in ANSI-C for Posix Systems, http://code.google.com/p/redis/
53. Project Voldemort – A Distributed Database, http://project-voldemort.com/

EXRT: Towards a Simple Benchmark for XML Readiness Testing

Michael J. Carey[1], Ling Ling[1], Matthias Nicola[2], and Lin Shao[1]

[1] Computer Science Department, University of California-Irvine, Irvine, CA 92697
[2] IBM Silicon Valley Lab, 555 Bailey Avenue, F270, San Jose, CA 95141
{lling1,lshao1,mjcarey}@uci.edu, mnicola@us.ibm.com

Abstract. As we approach the ten-year anniversary of the first working draft of the XQuery language, one finds XML storage and query support in a number of commercial database systems. For many XML use cases, database vendors now recommend storing and indexing XML natively and using XQuery or SQL/XML to query and update XML directly. If the complexity of the XML data allows, shredding and reconstructing XML to/from relational tables is still an alternative as well, and might in fact outperform native XML processing. In this paper we report on an effort to evaluate these basic XML data management trade-offs for current commercial systems. We describe EXRT (Experimental XML Readiness Test), a simple micro-benchmark that methodically evaluates the impact of query characteristics on the comparison of shredded and native XML. We describe our experiences and preliminary results from EXRT'ing pressure on the XML data management facilities offered by two relational databases and one XML database system.

Keywords: Database management, benchmarks, XML, XQuery, SQL/XML.

1 Introduction

Since the late 1990's, XML has been steadily gaining traction in the commercial IT and Web worlds. Perhaps more important than its initial document-centric motivation, the separation of (semantic) document markup and presentation, XML has become a widely used standard for Web-based information exchange between businesses, government agencies, and other organizations and applications. Numerous XML-based standards have been developed – in areas such as health care, finance, insurance, and virtually all facets of government – to specify the content of data and messages being exchanged. XML also serves as the foundation for the Web service and Service-Oriented Architecture (SOA) facilities that provide the information and operational fabrics that tie together most modern enterprise IT systems, both within and across enterprises. As a result, a large volume of XML data is being created on a daily basis. XML is also gaining traction as a semi-structured information model, i.e., as a way of handling data that is too variable for convenient relational storage.

Given the growth of XML for data sharing and an increasing focus on auditing and compliance, there is a strong need to manage and query significant quantities of XML data. Additionally, applications are starting to use XML as a database format as well;

R. Nambiar and M. Poess (Eds.): TPCTC 2010, LNCS 6417, pp. 93–109, 2011.

reasons include simpler (more flexible) database design plus potential performance benefits and simplified application development compared to converting XML to and from relational form. In anticipation of these needs, the W3C formed an XML Query working group in 1999, ultimately leading to XQuery 1.0 in 2007 [27]. Also, the SQL community added XML extensions to SQL, called SQL/XML [7], first to enable publishing of XML from relational data and later to support direct storage and querying of XML data. Relational database vendors have added extensive XML capabilities to their products. XML storage and query capabilities are available in Oracle [13], IBM DB2 [15], Microsoft SQL Server [19], and Sybase [24]. In parallel, several XML-targeted databases have emerged, such as Marklogic [11], Software AG's Tamino [22], and EMC's xDB, formerly known as X-Hive [8].

XML data management in the first half of the past decade involved "shredding" document instances, either generically or in a schema-driven manner, for storage and retrieval from tables in relational databases [23]. The data was then queried with SQL and "published" (reconstructed) using features such as XML publishing functions in SQL/XML. In recent years, however, the XML capabilities of commercial databases have reached a maturity point where vendors often recommend storing XML natively and using XQuery or SQL/XML for queries and updates. One XML evangelist argues that it no longer makes sense to disassemble and reassemble XML messages and business documents as they flow through enterprises' SOA infrastructures just to persist and query them [12]. One reason is that real-world business records represented in XML are often too complex to be mapped to a relational schema in a sensible manner. For example, financial trading records represented in FpML [26] or FIXML [25] can be highly variable and require hundreds of tables if shredded to a normalized relational schema. In such cases, shredding is costly to develop and to maintain over time when XML structures evolve. Instead, it is claimed that XML indexing and query processing technologies have matured enough to compete very favorably with relational technology, making XQuery the "right answer" [9, 12].

In this paper we assess the current state of XML and XQuery support in commercial database systems. We describe EXRT (*E*xperimental *X*ML *R*eadiness *T*est), a simple micro-benchmark designed to methodically evaluate XML data management tradeoffs such as the impact of query characteristics on the relative performance of shredded versus native XML. We describe our benchmark and present our experiences and preliminary results from using EXRT. Section 2 reviews related benchmarking work and how EXRT differs, while Section 3 describes the benchmark itself, including its data characteristics, the benchmark queries and updates, and the intended operating conditions. Section 4 presents preliminary results for EXRT implementations on two commercial relational database systems and one XML database system. Section 5 summarizes the main results and discusses future work.

2 Related Benchmarking Work

Most XML data management benchmarking has focused on XML documents and exercising a variety of features of XQuery. In contrast, EXRT tries to capture and exercise only those XML storage and processing capabilities that are "core" features

for enterprise-oriented XML use cases – e.g., features that would likely be exercised in scenarios like SOA message archiving and querying. In the early days of the relational era, a micro-benchmark known as the Wisconsin Benchmark [6] helped potential adopters evaluate different database offerings and drove vendors to improve their products. The EXRT design was inspired by the Wisconsin model and its early usefulness, hence our adoption of a micro-benchmarking based approach.

Regarding previous benchmarks, the database research community has proposed various XQuery and XML database benchmarks, including XMach-1[3], XMark [20], XPathMark [10], XOO7 [4], XBench [28], MBench [18], and MemBeR [2]. The only industry-led benchmark proposal to date is the TPoX (Transaction Processing over XML) benchmark from IBM and Intel [15]. Among these benchmarks, some are predominantly application-oriented and scale in the number of documents, such as TPoX and XMach-1, while others are designed as abstract, single-document micro-benchmarks, such as MBench and MemBeR. XMark, XPathMark, and X007 are also single-document micro-benchmarks; they each use data and queries that represent (artificial) application scenarios and then try to exercise *all* relevant aspects of the XQuery and XPath languages in the manner of a micro-benchmark. Further discussion and comparison of existing benchmarks can be found in [1,14,15,21].

EXRT differs significantly from both existing application-oriented benchmarks and existing micro-benchmarks. XMach-1 and TPoX are application-level benchmarks that measure the throughput of a multi-user read/write workload and evaluate a system's overall performance. In contrast, EXRT is a single-user micro-benchmark that methodically compares the performance impact of various query characteristics, making it complementary to XMach-1 and TPoX. EXRT evaluates "feature strength" while TPoX evaluates "system strength" at an aggregate level in the spirit of TPC-B and TPC-C. EXRT uses a subset of the XML documents from TPoX. TPoX has three collections of documents, including complex FIXML messages [25] whose XML Schema cannot be reasonably mapped and shredded to a normalized relational schema. EXRT uses a single collection of documents (about customers and accounts) that can be shredded to a sensible relational schema and therefore allow a comparison of native versus relational XML storage.

EXRT also differs from the existing micro-benchmarks in several ways:

- Existing micro-benchmarks use a single XML document and scale in terms of the size of this document. EXRT uses many small XML documents and scales in the number of documents, which is the more realistic scenario in practice [17,11,15].
- Existing micro-benchmarks focus on exercising *all* features of the XPath and XQuery languages, while EXRT does not. EXRT defines a set of atomic tests to assess how the selectivity and "width" of a given query affect performance.
- Existing micro-benchmarks examine XML and XQuery only. In contrast, EXRT evaluates both native and shredded XML storage with XQuery and SQL/XML.
- Results from the existing micro-benchmarks are useful predominantly for the designers of XQuery engines. EXRT provides additional information that is useful for designers of database schemas and applications that leverage both XQuery and SQL/XML functionalities [29,17].

3 The EXRT Benchmark Design

The EXRT benchmark focuses on data-oriented XML use cases rather than content- or document-oriented uses of XML. As a result, EXRT uses XML data for which a corresponding schema is available and knowledge of its query and update workloads is exploitable for creating suitable indexes. The EXRT workload is designed to exercise a range of query and update features that we believe have a high likelihood of being exercised by typical SOA or Web applications interacting with XML business data. Being a micro-benchmark and independent of a particular application, EXRT takes a neutral stance with respect to expectations about short- and long-term locality of data access. As a consequence, both "cold" and "hot" operation times are measured and reported, similar to what was done in OO7 [5]. For systems that support both XQuery and SQL/XML, both are tested to examine their relative performance.

It is not the goal of EXRT to determine whether one database system is superior to another. Hence, EXRT does not define a single aggregate performance metric that could be used to declare one implementation as a winner over another. Instead, EXRT produces a collection of response time results to study performance tradeoffs. The EXRT design is detailed in the remainder of this section.

3.1 EXRT Database Design

To support a controllable set of queries and updates, our initial thought was to follow the idea of the Wisconsin benchmark and design a generic synthetic (XML) database with randomly generated data values for EXRT. However, after examining existing XML benchmarks and their data, we decided to forgo reinvention of the wheel. We decided to utilize data from the open source TPoX benchmark [15]. We chose the TPoX data because we found its design to contain a sufficient range of XML features and "knobs" for EXRT's needs, including simple and complex data types, single- and multi-valued nesting of XML elements, and a selection of unique- and non-unique-valued attributes and elements for indexing and querying purposes.

The TPoX database models a hypothetical brokerage house. It has three XML document types, *CustAcc* (for customers and accounts), *Order* (for buy/sell orders), and *Security* (for securities). The *Order* schema uses the FIXML industry standard [25] while the other schemas are based on experience with real XML data in the financial sector. We chose the *CustAcc* schema for EXRT, as it contains key features we wanted and is still simple enough (in contrast to FIXML!) to explore both schema-specific shredded relational storage and native XML storage. We used the TPoX data generator and extended the TPoX workload driver, which is a configurable Java application to execute, randomize, and measure a user-defined database workload.

In TPoX, each customer has an associated *CustAcc* document that contains the customer information, account details, and holdings information for the customer. *CustAcc* documents range from 4KB to 20KB in size. (Schema and sample data are available on the Web [16].) The information stored for each customer includes:

- their name, with multiple short names and middle names possible
- their date of birth, gender, nationality, country, languages spoken, preferences, etc.
- their addresses, including all details and a set of phone numbers for each address

- their accounts, with a variety of account properties as well as their balances, set of value dates, and details of the holdings that make up the account.

For the XML storage case, *CustAcc* instances are stored "as is", either in a table with a single XML column (for relational systems) or in an XML collection (for the XML database). For the case of schema-specific shredded storage, normalizing the *CustAcc* XML schema into as few tables as possible yields the 12 tables shown in Fig. 1. The parent/child (referential) relationships between the tables in Fig. 1 mirror the containment hierarchy of the XML schema. In all cases, we (path-)indexed primary and foreign keys as well as elements and attributes used in non-key-based selections.

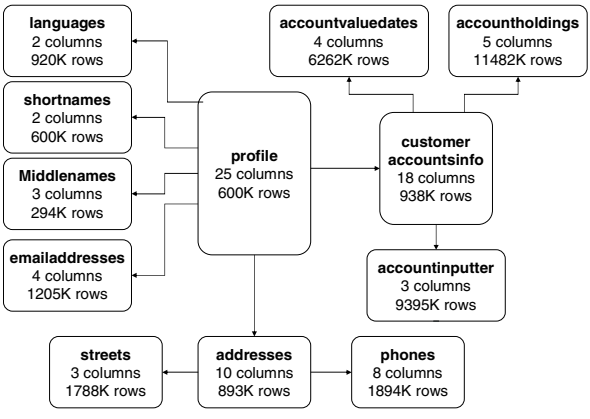

Fig. 1. Normalized Relational Schema for TPoX CustAcc Information

3.2 EXRT Queries and Updates

The EXRT operations listed in Table 1 include a set of queries that cover a range of behaviors that we expect in typical data XML use cases, plus a set of simple updates. For the case of native XML storage, EXRT tests both XQuery and SQL/XML versions of each query, if supported by the system under test. For shredded XML storage, EXRT performs SQL/XML with XML construction over the relational tables.

3.2.1 Queries

The first four queries in the EXRT benchmark are parameterized on customer ID. Each takes a range of customer IDs and extracts varying amounts of information about each customer. These queries test basic exact-match and range-based ID lookups as well as the cost of fetching and assembling the requested information. For native XML storage, their cost involves selection, navigation, element extraction, and the construction of new result documents. For shredded storage, their cost involves selection, joins, and result construction. The fraction of customer information fetched for each customer is varied from just basic information (touching only one table in the shredded case) up to all of the information (touching all tables in the shredded case). We quantify the fraction of information fetched by counting the number of shredded

tables involved and refer to this metric as the *width* of a query. We also vary the size of the customer id range requested to control the result cardinality of the query, which we refer to as a query's *height* (or *selectivity*).

Table 1. EXRT Benchmark Operations

Op	Description	Width
Q1	For given customer IDs, fetch the customers' minimal profile – consisting of their customer ID, title, first and last names, and suffix.	1
Q2	For given customer IDs, fetch the customers' basic profile – adding middle and short names and languages to the minimal profile.	4
Q3	For given customer IDs, fetch the customers' complete profile – adding e-mail information, addresses, streets, and phones to the basic profile information.	8
Q4	For given customer IDs, fetch all of the customers' information, including all of the information about their accounts and holdings.	12
Q5	For given customer IDs, get the complete information for all of their accounts.	5
Q6	For a given account ID, get the complete account information.	4
Q7	Given an account ID, get all of the customer information for the account's owner.	12
Q8	Get the average number of accounts for customers of a given nationality.	1
Q9	Given a country name and a tax rate, return the average account balance for customers in that country who have a tax rate greater than the specified rate.	3
I	Given an XML string containing all of the information for a new customer, insert the new customer into the database.	12
D	Given a customer ID, delete all information about this customer and his accounts.	12
NI1	*Node Insert 1:* Given a customer ID and an XML string with a new Address element, add the new address to the specified customer.	3
NI2	*Node Insert 2:* Given a customer ID and XML strings with a new Address element and a new e-mail address, add both to the specified customer.	4
NI3	*Node Insert 3:* Given a customer ID and XML strings containing a new Address element, a new e-mail address, and a new account, add these to the customer.	8
ND1	*Node Delete 1:* Given a customer ID plus an integer positional indication (1, 2, or 3), delete the indicated Address node from the customer's list of addresses.	3
ND2	*Node Delete 2:* Given a customer ID plus positional indicators for their address and e-mail lists, delete the indicated Address and Email nodes.	4
ND2	*Node Delete 3:* Given a customer ID, an account ID, and positional indicators for their address and e-mail lists, delete the indicated Account, Address, and Email.	8
NU1	*Node Update 1:* For a customer ID, update the customer's last contact date.	1
NU2	*Node Update 2:* Given a customer ID, a contact date, and the name of a new account officer, update the last contact date, upgrade the customer to premium status (premium = 'yes'), and update the assigned account officer's name.	2
NU3	*Node Update 3:* Given a customer ID, a contact date, the name of a new account officer, and an XML string with a list of addresses, update the customer's last contact date, upgrade the customer to premium status, update the account officer's name, and replace the customer's current list of addresses with the new list.	5

EXRT includes two versions of Q4, one that simply returns a whole customer document "as-is" and one that first extracts *all* nodes and values and then reconstructs the same customer document. Although the full reconstruction is not generally recommended, it is included to demonstrate the cost of XML (re)construction and highlight the performance benefit of storing and retrieving XML documents intact.

The next three EXRT queries (Q5, Q6, Q7) test the performance of navigation, path indexing, and document fragment extraction without reconstruction for complex objects (i.e., objects with nested sets of other objects). The widths of these queries are fixed by their information content, but we vary their selectivity. Comparing these three queries, notice that Query 5 does ID-based selection of a parent object and returns its nested child objects of a certain kind. In contrast, Query 6 performs ID-based selection of a nested object and then returns the selected nested object itself. Query 7 is an ID-based selection of a nested object but returns its parent object.

The last pair of queries (Q8, Q9) are simple summary queries with multiple predicates to spot-test the XML data aggregation capabilities of a system.

3.2.2 Update Operations

EXRT also contains basic insert, update, and delete operations, as listed in Table 1. EXRT includes full document insert and delete operations as well as node-level changes such as inserting, updating, or deleting individual pieces of an XML document. These operations are each applied to a randomly selected customer, changing either the customer information or nested information within the customer.

3.3 EXRT Testing Procedure

To provide performance information that is useful to XML application developers, we measure both "cold" and "hot" execution times for each query. The cold tests are performed by clearing the buffer pool before each query's execution, and different query parameters are randomly selected for each query execution. The buffer-clearing procedure used is vendor-dependent. Each of the relational database system vendors instructed us on how to ensure coldness for data and index pages. The XML-only database vendor actually added a cache-clearing feature for our benefit (which they plan to ship for general customer use in a future release). Despite assistance, we found the problem of ensuring a comparable degree of "coldness" across different systems to be challenging, especially for insert, update, and delete tests.

Our hot tests execute each query repeatedly with the same parameter values without clearing the buffer pool. In real-world applications, where data volumes are often much larger than main memory, systems require *some* physical I/O since they are never entirely hot nor entirely cold. Hot and cold tests represent the best and worst case scenarios and thus provide lower and upper bounds on the expected performance.

For operations that are able to take advantage of parameterized query APIs with a separation of query preparation and query execution, the reported times include only the execution time plus the time to fetch all the query results from the database server. To obtain stable results, each query or update is repeated 10 times. The measured elapsed times of the first execution as well the slowest and fastest of the remaining 9 executions are discarded. The resulting 7 measurements are then averaged. Queries are run sequentially (single-user) on an otherwise unloaded system.

The EXRT workload driver uses JDBC to interact with the relational systems and uses a roughly comparable Java API for the XML database system. Given the API rules and restrictions for the various systems and languages, the EXRT queries used parameter markers (with separate query preparation and execution steps), except for the following cases where literal string substitution was used instead: node updates in

RDB 1 and XQuery queries in RDB 2. RDB 1's JDBC API does not support pure XQuery, so a thin SQL/XML wrapper was needed for all of its queries.

The parameter values themselves, including any XML arguments (e.g., new customer accounts), were produced ahead of time as part of the database generation process, and the resulting parameter files were used to drive the benchmark.

4 Preliminary Results

Our tests were run on a Dell desktop system with a dual-core 3.16 GHz Intel® Core™ 2 Duo E8500 CPU, 4GB of main memory, and a pair of 320 GB 7200 RPM Western Digital disks. The operating system was Red Hat Enterprise Linux Client release 5.4 (Tikanga). The database instances for each system were created on a striped Linux file system volume in order to utilize both disks. All database files and logs were stored on that volume as well. We refer to the two relational systems as *RDB 1* and *RDB 2* and to the XML database system as *XDB*.

We strive to show each system in a good (fair) light and to run the systems with as comparable configurations as possible. It is not our goal to "race" the systems against each other, but to show both intra- and inter-system performance trends and tradeoffs.

For RDB 1, we used generally available code plus a generally available product patch to address JDBC driver issues with XML result sets. RDB 1's vendor also requested that a few queries be slightly rewritten to help the optimizer to find more efficient indexed access plans. Finally, due to an issue with the collection of binary double index statistics in RDB 1, we dropped the binary double index statistics as part of the EXRT database setup in order to prevent suboptimal access plans from being chosen. As EXRT managed to reveal these issues in shipping software, EXRT has already provided significant vendor value in this regard. For RDB 2 we used its generally available code and executed all queries "as is" without manual rewrites. For XDB we also used generally available code plus a cache-clearing option that was needed to obtain cold results.

4.1 Queries with XML Construction or Full Document Retrieval

In the following, all results are arranged in the same format, i.e. three charts in a row to represent RDB 1, RDB 2, and XDB, respectively. The legends in the charts mean:

- XQuery on XML = pure XQuery over an XML column or collection
- SQL/XML on XML = SQL/XML with embedded XQuery over an XML column
- SQL/XML on Rel = SQL over relational tables containing shredded XML. (Queries Q1 through Q7 use SQL/XML construction functions to produce XML results equivalent to the XQuery and SQL/XML results on XML columns.)

Note that query "Q4(re)" retrieves a full customer document by reading all of its values and reconstructing it, which is expected to be expensive. In contrast, "Q4" reads the full document in one piece, which is the preferred and more efficient operation for full document retrieval.

Fig. 2 shows the response times in milliseconds of the first four queries for a selectivity of 1 row. As expected, the performance on a hot system is much better than on a

cold system, for all queries and all systems. Constructing XML becomes increasingly more expensive for the two relational systems as the size and width of the result become bigger (Q1->Q2->Q3->Q4(re)); this effect is not seen for XDB in the 1-row case. In RDB 1 and RDB 2, constructing XML from cold relational tables becomes more expensive with increasing width because the number of I/Os made to distinct index and data pages increases with the width, i.e. with the number of tables accessed and joined. Retrieving a stored XML document as a single unit (Q4) is much faster than reconstructing a document from its individual nodes (Q4(re)), which clearly shows the benefit of a native XML column over shredded relational XML storage.

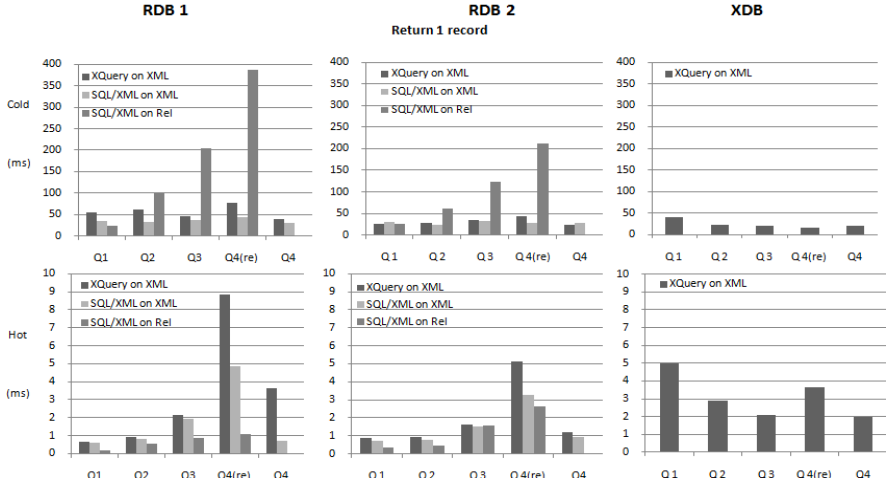

Fig. 2. Query Response Times in Milliseconds (Result set size = 1)

The hot tests show that constructing XML from shredded relational data can sometimes outperform all native XML options, i.e. native XML in RDB 1, RDB 2, and XDB. This is most evident in RDB 1 as well as in RDB 2 when the system is not cold and the width of the queries is low (Q1 and Q2), i.e. when only few of the tables need to be joined and the amount of constructed XML per result row is small.

Some of the hot results can be improved if queries are stored as precompiled stored procedures, or "modules" in the case of XDB, on the database server. However, our goal was to test a scenario with dynamically submitted queries. As a result, both RDB 2 and XDB submitted XQuery without using a separate "prepare" step in the application to compile the queries before submitting them to the database.

Fig. 3 shows the results for Q1-Q4 when the predicates select 60 customers instead of just one document (row). For a larger selectivity the elapsed time increases, as expected, and so does the effect of the width on query performance. Constructing XML documents with a large number of elements (e.g. Q3 and Q4(re)) then performs best in XDB. However, retrieving full XML documents without reconstructing them (Q4) performs slightly better in RDB 2 than XDB in both hot and cold systems.

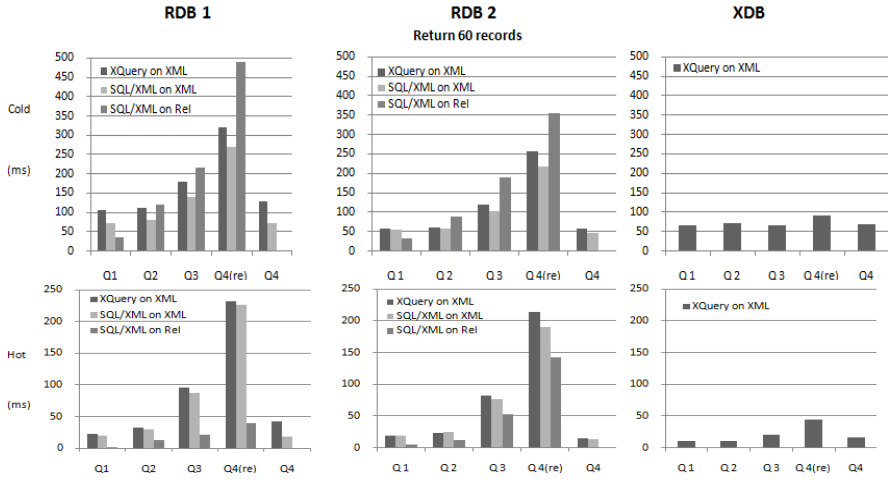

Fig. 3. Query Response Times in Milliseconds (Result set size = 60)

As the result set size increases to 600 rows in Fig. 4, the difference between hot and cold performance begins to diminish, because two other factors begin to outweigh the I/O cost. One factor is the CPU cost to construct XML for many result rows, and the other is the overhead of shipping results from the database server to the client. This trend became even stronger for a result set of 6000 records (charts omitted for brevity). In Fig. 4, the XQuery elapsed time for the cold run in RDB 1 is 4156 ms, slightly exceeding the scale of the chart, as indicated by the arrow. The performance benefit of Q4 over Q4(re) clearly shows the benefit of native XML columns in relational databases over shredded relational storage.

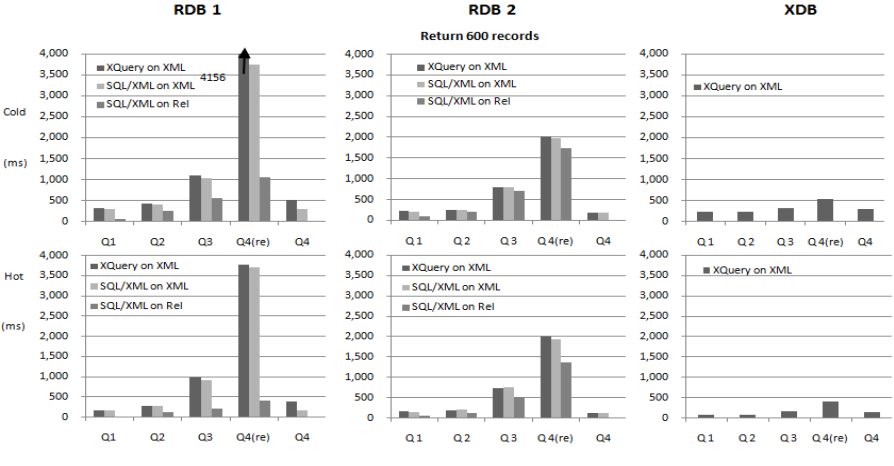

Fig. 4. Query Response Times in Milliseconds (Result set size = 600)

4.2 Queries that Extract Partial Documents

Query 5 extracts "account" fragments from customer documents that match given customer IDs, while Query 6 extracts "account" fragments that match given account IDs. For comparison, Query 7 returns full customer documents that include a given account ID. In XQuery and SQL/XML on XML columns, these three queries return XML documents or fragments *without* constructing new XML elements. Only the SQL/XML queries over relational tables require XML construction in this case. This explains their higher elapsed time than, e.g., SQL/XML over native XML columns.

Fig. 5 shows the response times of Q5-Q7 when their predicates match only one document. In RDB 1, a suboptimal execution plan for the XQuery implementation of Q6 leads to a high response time. In the hot tests with RDB 2, the difference between XQuery versus SQL/XML access to XML columns is negligible, less than 0.1 msec. This is expected since equivalent XQuery and SQL/XML queries are compiled into identical execution plans in RDB 2. In RDB 2, Q7 on relational tables can actually be significantly faster than shown in Fig. 5 (similar to Q5) if textual parameter values are used rather than parameter markers, allowing the optimizer to estimate the selectivity of the range predicates more accurately. Reconstructing the full documents from relational tables is not vendor-recommended for RDB 2, though, as both XQuery and SQL/XML over XML columns provide better performance.

In the cold tests, XDB performs similar to or slightly better than RDB 1 and RDB 2 for Q5-Q7. In the hot runs, however, SQL/XML on XML columns in RDB 1 and RDB 2 as well as XQuery in RDB 2 outperform XDB.

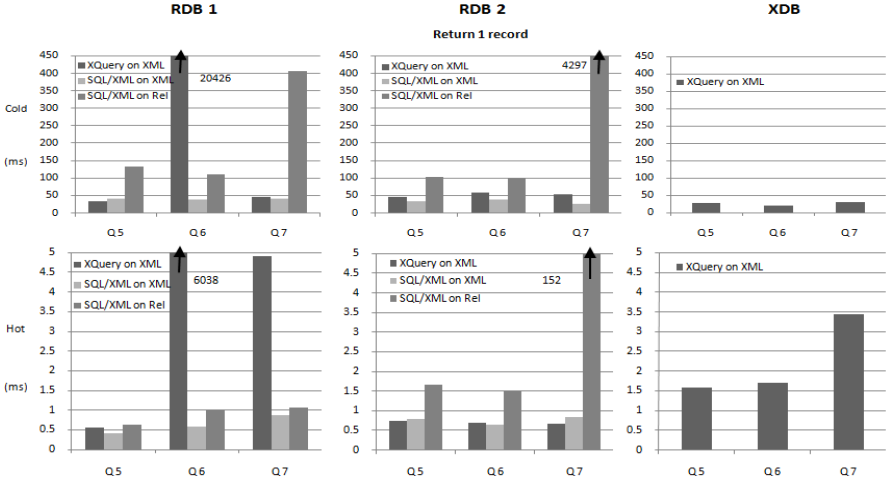

Fig. 5. Query Response Times in Milliseconds (Result set size = 1)

Fig. 6 and Fig. 7 show the elapsed times of the same three queries when their predicates select 60 and 600 results, respectively. The larger selectivity means that more documents need to be processed, which amplifies most of the trends already observed in Fig. 5. In particular, the overhead of SQL/XML on shredded relational XML storage increases for RDB 1 and RDB 2, which again underscores the value of

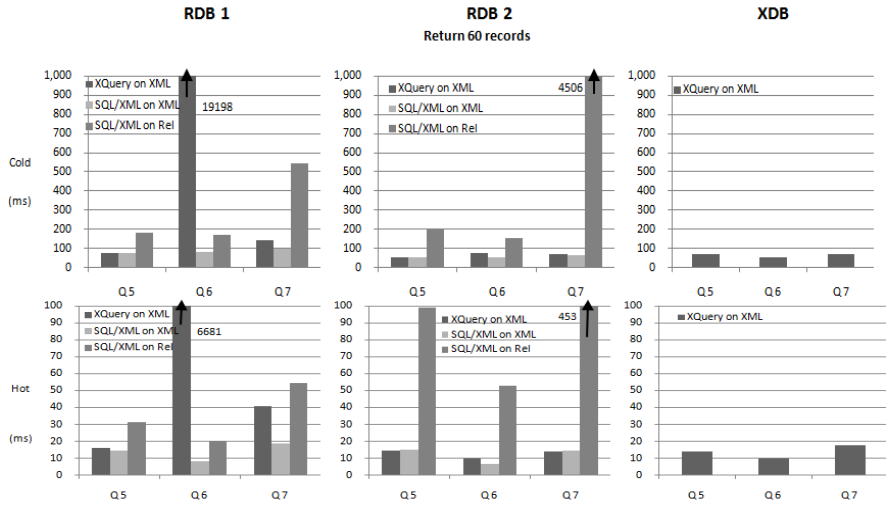

Fig. 6. Query Response Times in Milliseconds (Result set size = 60)

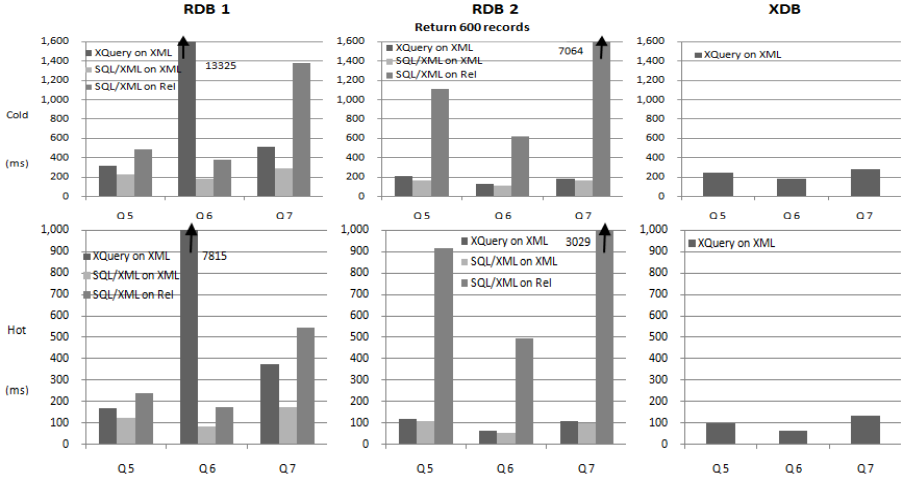

Fig. 7. Query Response Times in Milliseconds (Result set size = 600)

the native XML database capabilities. The XQuery performance in RDB 2 is similar to the performance of XDB.

4.3 Aggregation Queries

These queries are basic analytical queries with several predicates for selection and aggregation of a metric of interest. Given the predicates in these queries, both queries require access to many documents that cannot possibly be clustered by customer ID. Hence, random access into the document collection is critical for good performance, especially when a system's memory is not entirely hot. In a cold environment, where

physical I/O to fetch data from secondary storage is necessary, SQL on relational tables in RDB 1 and RDB 2 outperforms XDB (Fig. 8, top). Since Q8 and Q9 are aggregation queries that return a simple numeric result, RDB 1 and RDB 2 use regular SQL without XML construction or embedded XQuery. In a perfectly hot environment, where all data resides in main memory and the queries run entirely CPU-bound, RDB 1 performs best in Q8 while XDB performs best in Q9 (Fig. 8, bottom). As indicated in the charts for RDB 1 in Fig. 8, Q9 is a query for which inaccurate binary DOUBLE index statistics had to be removed and an additional binary DOUBLE cast expression was added to the query to assist RDB 1's optimizer.

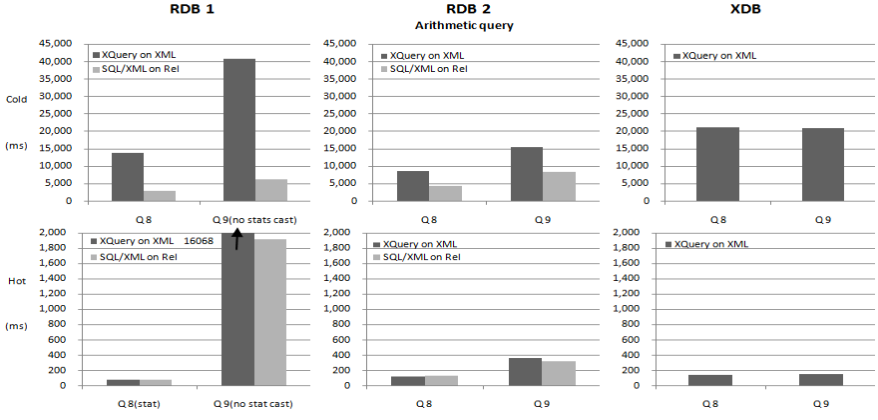

Fig. 8. Query Response Times in Milliseconds – Aggregation queries

4.4 Full Document Inserts and Deletes

The full documents inserts and deletes as well as the document updates discussed in section 4.5 were each executed as a *single* operation on a cold system. The reason for this was the desire to have short tests that match EXRT's micro-benchmark nature while capturing the I/O costs incurred for reading and possibly for writing data and index pages that need to be modified. However, we found it difficult to define a test procedure that makes such atomic write experiments fully comparable across multiple different database management systems. Hence, we consider the tests and results in sections 4.4 and 4.5 preliminary and subject to further refinement. For example, some of the vendors, including XDB, agreed that a more comparable test would execute many thousands (or perhaps even millions) of insert, update, or delete operations in sequence to capture the *steady state performance* of each system. We take this as an area for future work.

With this caveat, Fig. 9 shows the response time for inserting and deleting the information for a *single* customer. In the relational systems, these operations are significantly less expensive on an XML column than on shredded XML. The reason is that shredded XML requires row manipulation for 12 separate tables and indexes (using regular SQL DML statements). In XDB, the insert and delete times were significantly lower than in the relational systems, depending on how the system was prepared for the cold write operation.

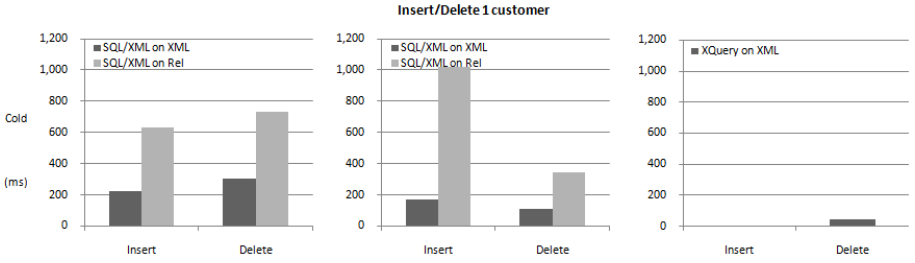

Fig. 9. Inserting/Deleting a single Customer Document

4.5 Sub-Document Insert, Delete, and Update Operations

Fig. 10 shows the response times for a single cold update transactions that inserts, deletes, or updates nodes within an XML document. When XML is stored in relational tables, the node insert and node updates require incoming XML fragments to be parsed and translated into regular SQL insert and update statements (without XML functions) on multiple relational tables. For RDB 1, this tends to be less expensive than performing an XML write operation on a single XML document in a single table–except for test cases with a high width, such as NI3 and ND3 which have width 8. The opposite is true for RDB 2, where XML updates on an XML column

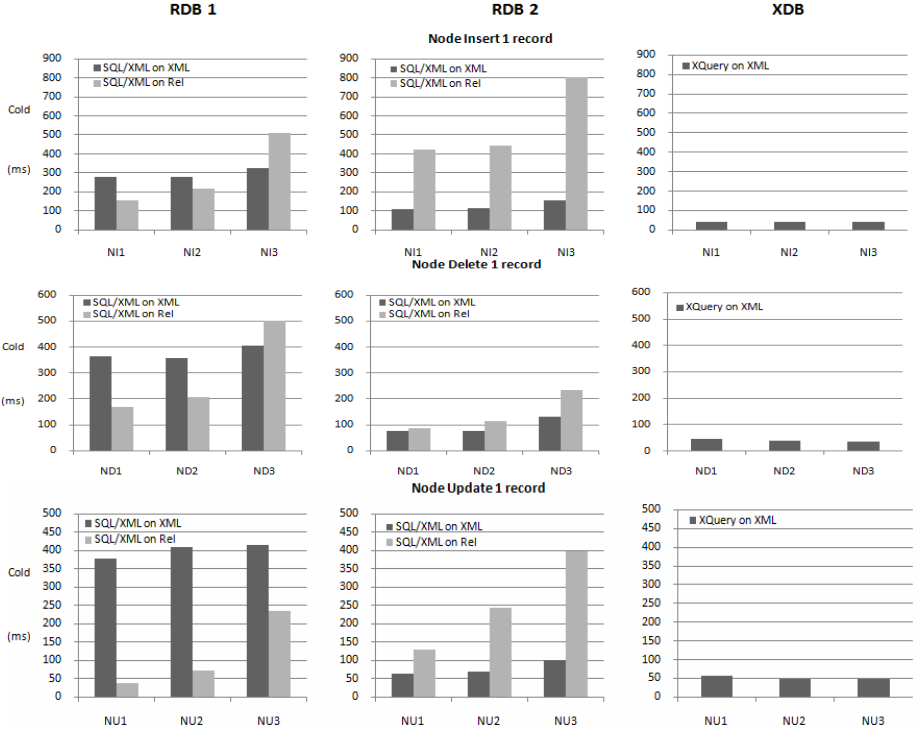

Fig. 10. Inserting/Deleting/Updating Nodes in one Existing XML Document

always outperform the conversion of incoming XML snippets into relational inserts and updates. The results for RDB 1 and RDB 2 indicate that the cost of document manipulation increases with the width, and more so for shredded storage than for XML columns. In contrast, the width does not seem to affect the sub-document insert, delete, and update operations in XDB.

5 Conclusions and Future Work

In this paper, we have described the design of EXRT (*E*xperimental *X*ML *R*eadiness *T*est), a new micro-benchmark for (i) evaluating XML data management tradeoffs such as the impact of query characteristics on the relative performance of shredded versus native XML and (ii) assessing the robustness of the XML storage and query support in commercial database systems. In addition, we demonstrated the use of EXRT to benchmark the XML data management capabilities of two commercial relational database systems as well as a commercial XML database system. We observed some similarities and some significant differences between the performance behavior of the three systems and discovered issues related to optimizations for XQuery versus SQL/XML queries over XML columns for one of the systems.

The performance of shredded XML versus natively stored XML is query-dependent, so the right tradeoff for a given application depends on its target use case(s). Shredded XML is akin to columnar storage for traditional relational data, and the tradeoff depends heavily on how much of a record's full "width" has to be touched. Queries that touch a low width of data can run faster with SQL/XML on relational tables than on XML columns in relational systems or an XML collection in the XML database. However, shredding XML documents into a relational schema in the first place is easily 3 to 5 times more expensive than inserting it natively into XML columns.

There are several potential avenues for future work based on EXRT. One interesting pursuit would be to test the performance of a broader set of commercial systems, as we have only scratched the surface with the three systems considered here. We plan to make the EXRT benchmark package available to others via the web, most likely by providing a reference implementation for one of the relational systems. Another useful pursuit would be to refine the test procedure for inserts, updates, and deletes in EXRT. A final interesting avenue would be to develop an EXRT-like micro-benchmark that targets the data characteristics and common operations for content-oriented XML use cases such as large-scale document management and automated publishing.

Acknowledgments. The authors would like to thank our engineering contacts from each of the three vendors whose products we benchmarked for their reviewing input, sanity checking, and advice related to our choices of server configurations, our physical database setups, and the performance results. (You know who you are!).

References

1. Afanasiev, L., Marx, M.: An Analysis of XQuery Benchmarks. Inf. Syst. 33(2) (2008)
2. Afanasiev, L., Manolescu, I., Michiels, P.: MemBeR: A Micro-benchmark Repository for XQuery. In: XML Database Symposium, XSym (2005)

3. Böhme, T., et al.: Multi-user Evaluation of XML Data Management Systems with XMach-1. In: Bressan, S., Chaudhri, A.B., Li Lee, M., Yu, J.X., Lacroix, Z. (eds.) CAiSE 2002 and VLDB 2002. LNCS, vol. 2590, pp. 148–159. Springer, Heidelberg (2003)
4. Bressan, S., et al.: XOO7: Applying OO7 Benchmark to XML Query Processing Tools. In: International Conference on Information and Knowledge Management, CIKM (2001)
5. Carey, M., DeWitt, D., Naughton, J.: The 007 Benchmark. In: SIGMOD Conference (1993)
6. DeWitt, D.: The Wisconsin Benchmark: Past, Present, and Future. In: The Benchmark Handbook for Database and Transaction Systems, 2nd edn. Morgan Kaufman, San Francisco (1993)
7. Eisenberg, A., Melton, J.: Advancements in SQL/XML. SIGMOD Record 33(2) (2004)
8. EMC, XDB Product Details, https://community.emc.com/docs/DOC-3111
9. Florescu, D.: Personal communication (2007)
10. Franceschet, M.: XPathMark - An XPath benchmark for XMark Generated Data. In: XML Database Symposium, XSYM (2005)
11. Holstege, M.: Big, Fast, XQuery: Enabling Content Applications. IEEE Data Engineering Bulletin 31(4) (2008), http://sites.computer.org/debull/A08dec/marklogic.pdf
12. Malaika, S.: Universal Services for SOA, WOA, Cloud and XML Data. In: Data Services World 2008 (2008), http://www.dataservicesworld.com/read/dataservices.sanjose.malaika.v7.pdf
13. Murthy, R., et al.: Towards an enterprise XML architecture. In: SIGMOD 2005 (2005)
14. Nambiar, U., et al.: XML Benchmarks Put to the Test. In: 3rd Internat. Conf. on Information Integration and Web-based Applications & Services, IIWAS (2001)
15. Nicola, M., Kogan, I., Schiefer, B.: An XML Transaction Processing Benchmark. In: ACM SIGMOD International Conference on Management of Data (2007)
16. Nicola, M., van der Linden, B.: Native Support XML in DB2 Universal Database. In: 31st International Conference on Very Large Databases, VLDB 2005 (2005), http://tpox.sourceforge.net/
17. Nicola, M.: Lessons Learned from DB2 pureXML Applications A Practitioner's Perspective. In: Lee, M.L., Yu, J.X., Bellahsène, Z., Unland, R. (eds.) XSym 2010. LNCS, vol. 6309, pp. 88–102. Springer, Heidelberg (2010)
18. Runapongsa, K., et al.: The Michigan Benchmark: Towards XML Query Performance Diagnostics. In: Proceedings of the 29th VLDB Conference (2003)
19. Rys, M.: XML and Relational Database Management Systems: Inside Microsoft SQL Server. In: SIGMOD 2005 (2005)
20. Schmidt, A., et al.: XMark: A Benchmark for XML Data Management. In: International Conference on Very Large Data Bases (VLDB), pp. 974–985 (August 2002)
21. Schmidt, K., Bachle, S., Harder, T.: Benchmarking Performance-Critical Components in a Native XML Database System. In: International Workshop on Benchmarking of XML and Semantic Web Applications, BenchmarX 2009 (2009)
22. Schöning, H.: Tamino - A DBMS Designed for XML. In: ICDE 2001 (2001)
23. Shanmugasundaram, J., et al.: A General Technique for Querying XML Documents Using a Relational Database System. SIGMOD Record 30(3), 20–26 (2001)
24. Sybase: XML Services in Adaptive Server Enterprise, http://infocenter.sybase.com/help/topic/com.sybase.dc30020_1251/pdf/xmlb.pdf
25. The Financial Information eXchange Protocol (FIXML), http://www.fixprotocol.org/specifications/fix4.4fixml

26. The Financial Products Markup Language (FpML), http://www.fpml.org/
27. XQuery, http://www.w3.org/TR/xquery/
28. Yao, B., Özsu, M.T., Keenleyside, J.: XBench - A Family of Benchmarks for XML DBMSs. In: Bressan, S., Chaudhri, A.B., Li Lee, M., Yu, J.X., Lacroix, Z. (eds.) CAiSE 2002 and VLDB 2002. LNCS, vol. 2590, pp. 162–164. Springer, Heidelberg (2003)
29. Liu, Z.H., Murthy, R.: A Decade of XML Data Management: An Industrial Experience Report from Oracle. In: ICDE 2009, pp. 1351–1362 (2009)

Transaction Performance vs. Moore's Law: A Trend Analysis

Raghunath Nambiar[1] and Meikel Poess[2]

[1] Cisco Systems, Inc., 3800 Zanker Road, San Jose, CA 95134, USA
+1- 408-527-3052
rnambiar@cisco.com
[2] Oracle Corporation, 500 Oracle Pkwy, Redwood Shores, CA 94065, USA
+1-650-633-8012
meikel.poess@oracle.com

Abstract. Intel co-founder Gordon E. Moore postulated in his famous 1965 paper that the number of components in integrated circuits had doubled every year from their invention in 1958 until 1965, and then predicted that the trend would continue for at least ten years. Later, David House, an Intel colleague, after factoring in the increase in performance of transistors, concluded that integrated circuits would double in performance every 18 months. Despite this trend in microprocessor improvements, your favored text editor continues to take the same time to start and your PC takes pretty much the same time to reboot as it took 10 years ago. Can this observation be made on systems supporting the fundamental aspects of our information based economy, namely transaction processing systems?

For over two decades the Transaction Processing Performance Council (TPC) has been very successful in disseminating objective and verifiable performance data to the industry. During this period the TPC's flagship benchmark, TPC-C, which simulates Online Transaction Processing (OLTP) Systems has produced over 750 benchmark publications across a wide range of hardware and software platforms representing the evolution of transaction processing systems. TPC-C results have been published by over two dozen unique vendors and over a dozen database platforms, some of them exist, others went under or were acquired. But TPC-C survived. Using this large benchmark result set, we discuss a comparison of TPC-C performance and price-performance to Moore's Law.

Keywords: Trends in System and Database Performance, Benchmark Standards.

1 Introduction

Intel co-founder Gordon E. Moore described in his famous 1965 paper [1] that the number of components in integrated circuits had doubled every year from the invention of the integrated circuit in 1958 until 1965, and predicted that the trend would continue for at least ten years. Around 1970 Caltech professor, VLSI pioneer, and entrepreneur Carver Mead coined the term the "Moore's Law". Moore slightly altered his formulation of the law over time. In 1975, Moore refined his projection to a doubling every two years [2]. Later, David House, an Intel colleague, who factored in the

R. Nambiar and M. Poess (Eds.): TPCTC 2010, LNCS 6417, pp. 110–120, 2011.

increase in performance of transistors, concluded that integrated circuits would double in performance every 18 months. History has shown that the capabilities of many digital electronic devices, such as processors, memory, etc. are strongly linked to Moore's law. All of these are improving at exponential rates. His prediction has proven to be very accurate. Some suggest that this is because the law is used in the semiconductor industry to guide long-term planning and to set targets for research and development. Figure 1 shows the CPU transistor count starting from 1971 to 2008. The graph is also annotated with the major milestones in the history of the Intel processor product line.

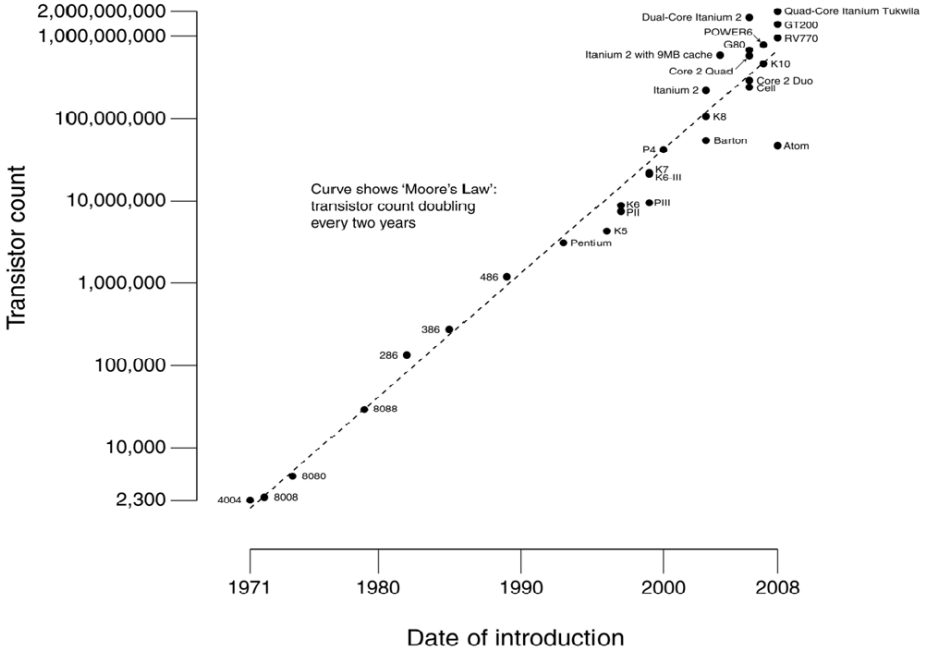

Fig. 1. Number of transistors on a single CPU (source Wikipedia.org)

But, does the exponential transistor growth, predicted by Moore's Law, also translate into exponentially greater performance of systems supporting the fundamental aspect of our information based economy, namely transaction processing systems? Many suggest it doesn't. Especially complex parallel applications that involve a diverse set of components are difficult to scale with processor performance because of their non-processor bottlenecks such as disks, software components, storage and connectivity.

In this paper we look at transaction processing systems, which are directly and indirectly involved in virtually every moment of our lives, e.g. phone calls, bank transactions, stock trades and flight reservations. As processor speeds have increased over the last two decades, performance of other critical components of transaction processing systems, such as disk latencies, I/O controllers and network latencies, have lagged far behind. Additionally, traditional algorithms may not be designed to scale to transaction

processing systems with a larger number of processors, large amount of memory and large number of disks. In this paper we look at performance improvements of transaction processing systems, not just one component but total system that includes database server, storage, connectivity and software including database management systems and application middle-tier.

In order to show relative performance gains of large complex application systems over time and to draw a comparison to Moore's Law, one needs consistent, verifiable performance data over a long period of time across a diverse set of platforms. The two most prominent industry standard benchmark organizations to publish benchmarks since the late 1980's are the Transaction Processing Performance Council (TPC), established 1987 and the Systems Performance Evaluation Corporation (SPEC), established 1988. The TPC's focus has been total system performance and price-performance under database workloads, including: server, storage, connectivity and software. All results have a price-performance metric audited by an independent TPC certified auditor. Like the TPC, the SPEC develops suites of benchmarks intended to measure system level performance. These suites are packaged with source code and tools and are extensively tested for portability across platforms before they are released. Unlike the TPC results, the SPEC results are peer audited. While the SPEC has been revising their benchmarks frequently the TPC has long lasting specifications. During its over 20 year long history the TPC has created and maintained significant benchmarks, such as TPC-A, TPC-B, TPC-C, TPC-H, TPC-W and TPC-E. Its flagship OLTP benchmark, the TPC-C, was first introduced in June 1992. Since then it has undergone several modifications. In its two decades of existence there have been over 750 results on dozens of hardware and software platforms. All major hardware and database vendors of yesterday and today have published TPC-C benchmarks, some of them are in business while others went out of business or were acquired. TPC-C survived, tracking the evolution of processor architectures (MIPS, RISC, CISC etc.), server architectures (rack mounted, SMP, clusters, blades etc.) and database technologies. There is no single benchmark standard that comes close at claiming such industry acceptance and life span. These factors make TPC-C the ideal candidate for conducting a performance and price-performance trend analysis that compares one of the most important factors that touch every second of our lives in the information era, namely transaction performance to Moore's law.

The remainder of this paper is organized as follows. Section 2 gives a brief overview of those parts of the TPC-C benchmark specification that are necessary to understand the subsequent sections. It also includes a history of the different revisions of TPC-C and explains why results from its different revisions can be used for the purpose of our comparison to Moore's Law. Section 3 demonstrates the performance and price-performance trends of TPC-C results from 1993 to 2010 and compares these trends to Moore's predictions.

2 Background of TPC's OLTP Benchmarks and Why TPC-C Is Suitable for Long Term Trend Analysis

TPC's first OLTP benchmark specification, TPC-A, was published in November 1989. Built upon Jim Gray's DebitCredit benchmark TPC-A formalized the rules, which all vendors had to obey in order to publish a benchmark result. About one year

later, TPC-B was developed. TPC-B was a modification of TPC-A, using the same transaction type (banking transaction) but eliminating the network and user interaction components. The result was a batch transaction processing benchmark. After two years of development, in June 1992, TPC's third OLTP benchmark specification, TPC-C, was approved.

Most TPC-C systems are implemented in three tiers mimicking typical transaction processing systems. A TPC-C benchmark implementation is also referred to as the System Under Test (SUT):

1. Tier: Driver System
2. Tier: Client
3. Tier: Database Server

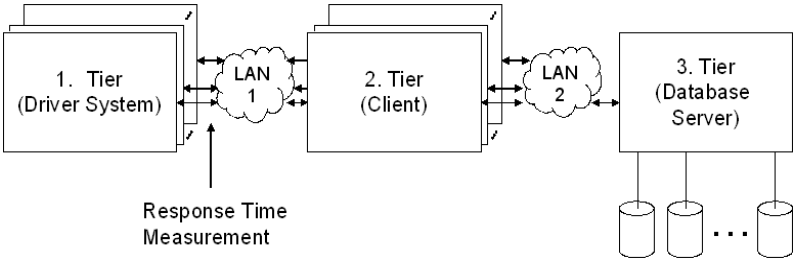

Fig. 2. Typical TPC-C System Setup, SUT (conceptual)

Compared to previous OLTP benchmarks, the TPC-C has a more representative database schema, several complex transactions exercising fundamental database functionalities and strict implementation requirements. It simulates a complete computing environment where a population of users executes transactions against a database. These transactions include entering and delivering orders, recording payments, checking the status of orders, and monitoring warehouse stock levels. In order to maintain TPC-C's applicability to systems of differing capacity, TPC-C implementations must scale both the number of users and the size of the database proportionally to the computing power of the system to be measured. The unit used in TPC-C to measure performance specifies the number of processed new-order transactions per minute (tpmC) while fulfilling the rest of the TPC-C transaction mix workload. There is response time threshold for each transaction: 90% of each type of transaction must have a response time of at most 5 seconds, except stock-level, which is allowed to be at most 20 seconds. TPC price/performance metric ($/tpmC) is the ratio of 3 year cost of ownership to tpmC. In addition to performance, dependability aspects of a fully production-ready system are also tested, by requiring full ACID properties (Atomicity, Consistency, Isolation and Durability).

2.1 History of TPC-C Revisions

In the first 18 months after approval TPC-C underwent two major revisions (Revision 2 and 3). Both revisions were comparable due to their minimal effect on existing

results. Revision 4 failed to get the necessary support from TPC members. In October 2000 Revision 5 of TPC-C was approved. This revision contained substantial changes with a large impact on existing results, which made it non-comparable to previous revisions under the rigid TPC rules. Since benchmark sponsors[1] wanted to protect their investments in the large set of results that were already published across many hardware and software platforms, the TPC created a set of requirements and guidelines that allowed a onetime upgrade of results from Revision 3 to Revision 5. The following section highlights the changes in the different revisions and reasons why the results can be compared.

Revision 2.0, released October 20th 1993

Revision 2.0 included additional wording in the General Implementation Guideline section. This section defines the requirement of a benchmark implementation and specifically disallows "benchmark specials". Although obvious to the TPC, rules to explicitly disallow hardware and software that was written specifically and solely to run TPC-C was not included in the TPC-C specification until this major revision. Additionally new disclosure requirements including a numerical quantity summary was added to the specification. Revision 2.0 requires the most relevant numerical data to be included as part of the executive summary of the benchmark publications and not scattered in the Full Disclosure Report.

Revision 3.0, released February 15th 1995

Revision 3.0 added new transaction monitor requirements for routing transactions. The first results published under revisions 1 and 2 revealed that only limited transaction monitor functions were exercised. In order to avoid unrealistic implementations of these functions in the operating system, new requirements were added that require a full featured transaction monitor in case such functions were used. As it became apparent that the distinction between Wide Area Networks and Local Area Networks technologies in the context of TPC-C was becoming fuzzy, the distinction between the two was removed. Careful examination of results revealed that vendors were pricing the lowest cost terminal available on the market. However, terminals accounted for a substantial portion of the reported price. As a consequence, terminals were excluded from the pricing to refocus the price on more crucial components of the configuration.

Revision 5.0, released October 18th 2000

The changes in Revision 5 can be divided in pricing related changes and benchmark run rule related changes. The pricing related changes had no impact on the transactions-per-minute metric (tpmC), but changed the total cost of ownership (TCO) and hence impacted the price-performance metric. They reduced the maintenance support period from five to three years and increased the weekly uptime requirements from eight hours, Monday through Friday to 24-hours seven days a week. The changes also removed the requirement to price the terminal connectivity network (hubs, switches),

[1] A benchmark sponsor is a company that publishes a TPC benchmark result.

which refocuses the benchmark on the client-tier, database-tier and their connectivity. They also allowed pricing quotes from web pages and print materials, which accommodates growing acceptance of online and direct purchase models. This changed the source of pricing, but had no impact on actual pricing. The last pricing related change was the reduction of the disk space requirements from 180 to 60 days. This is the disk space sufficient to store and maintain the data generated during a period of 60 days (instead of 180 days) of activity with an average of 8 hours per day at the reported tpmC rate. The intention was to reduce the cost for running benchmarks, but it made the total cost of ownership non comparable to results from previous revision.

The run rule change had no impact on performance or on price-performance. It increased the measurement interval from 20 minutes to 2 hours ensuring that the system under test can sustain the reported tpmC and guaranteeing that modified database records are written to durable media every 30 minutes for two hours through check point mechanisms. The TPC allowed upgrading Revision 3 to Revision 5 results if benchmark sponsors re-priced their systems to accommodate the pricing changes explained above. The two increased measurement interval requirement was waved.

2.2 Why TPC-C Is a Good Candidate for Technology Trend Analysis

The previous section has shown that the TPC-C specification evolved to remain as representative as possible of current practice without fundamentally changing the workload and pricing model. There were changes on implementation requirements and pricing, but the benchmark fundamentals including data population, transactions, execution rules and metric remained unchanged. This makes the TPC-C workload the ideal candidate for performance and price-performance trend analysis.

The pricing changes impacted the overall price of a system. For example in Revision 2 the price of terminals accounted for a substantial portion of the reported price, reducing maintenance support pricing to 3 years down from 5 years, and reducing the disk space requirements to 60 days from 180 days. Although the impact of these changes to TPC-C performance and price-performance is low, the strict TPC rules do not allow comparisons of results across major revisions. This is because benchmark sponsors usually compete over single digit percent differences. For our analysis in the next sections we use the performance and price-performance metrics without any adjustments because the changes had very little impact from a historical trend perspective. First of all, the fundamentals of the benchmark did not change. Secondly, due to the significant improvements in performance while cost of almost every component dropped, even the unadjusted performance metric and price-performance metric reflected overall industry trend. And finally, such changes as mandating commercially available transaction monitor and removing the terminal requirements are reflections of general changes in the industry. In the early 90's customers implemented their own home grown transaction monitors, which were eventually replaced by feature rich commercial TP monitors. In those days OLTP users accessed the servers over directly connected terminals, those were later replaced by remote web users.

3 Comparison of TPC-C Metrics with Moore's Law

In this section we analyze whether TPC-C performance improvements over the last 20 years agree with Moore's Law. We analyze both TPC-C's primary performance [tpmC] and price-performance [$/tpmC] metrics.

TPC-C benchmarks are published using a large range of system configuration sizes, which range from single server with one processor and a few disks to large cluster of servers with hundreds of processors and thousands of disk drives. Consequently, performance ranges from hundreds to millions tpmC. In order to compare the performance across various system sizes, we normalize the TPC-C performance metric by dividing the reported tpmC number by the number of processors (sockets) that are configured in the 3rd tier, i.e. the database server. We refer to this normalized metric as the NtpmC. For each year we then compute the average NtpmC for all TPC-C publications. Since the price-performance metric already accounts for the higher performance by an increased system cost, we do not need to adjust the price-performance with the number of processors.

3.1 Comparison of TPC-C Performance with Moore's Law

In this section we compare NtpmC, which is TPC-C's primary performance metric, normalized to the number of processors, to the adjusted[2] Moore's Law. The triangle markers in Figure 3 indicate the average Ntpmc per year from 1993 to 2010 on a logarithmic scale (base=10). The triangles are annotated with the major milestones in TPC-C publications. The dotted line shows TPC-C performance improvement using Moore's law as amended by David House, i.e. performance doubles every 18 months. As a starting point we use the average NtpmC of 1995, as the number of results available for 1993 and 1994 is too small to indicate a representative performance. We calculate the dotted line with the following function:

$$f(y) = f(y_0) * 2^{\frac{2}{3}*(y-y_0)} \quad (1)$$

$f(y)$ is the performance projection for year y and $f(y_0) = 426.65\ NtpmC$ is the base performance with $y_0 = 1995$.

Interestingly, the performance graph calculated using Moore's Law almost superimposes the data points taken from TPC-C publications [NtpmC]. There are three areas where NtpmC slightly over and under-performs Moore's Law.

In the years 1993 and 1994 and from years 1996 to 2000 Moore's Law slightly underestimates NtpmC. There are only a few results available for the first years, one result for 1993 and four results for 1994. Hence, the numbers obtained for these years are unlikely to be representative for performance improvements of technology in those years. The years between 1995 and 1999 saw a respectable number of benchmark results (between 21 and 76). Hence, the average of these results can be viewed as a very close representation of the performance that was achievable in these years.

[2] Adjusted as by David House: processor performance doubles every 18 months.

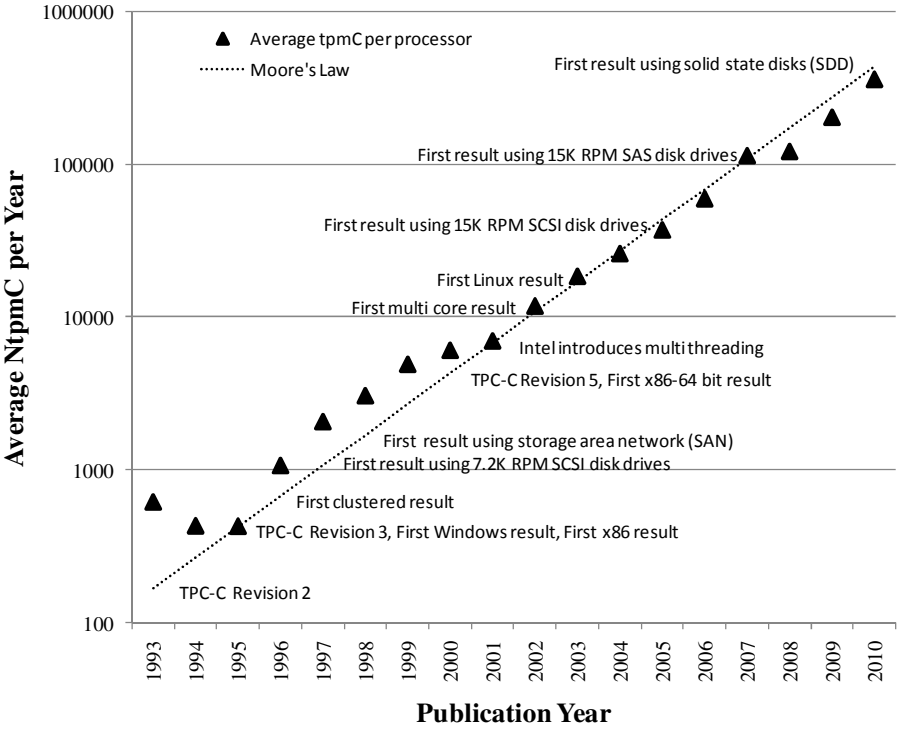

Fig. 3. TPC-C primary performance metric trend vs. Moore's Law

During the years 1996 to 2000, Moore's Law slightly underestimates NtpmC, while starting with 2008 it slightly overestimates NtpmC. In the period between 1996 and 2000 the publications used a very diverse combination of database and hardware platforms (about 12 hardware and 8 software platforms). Between 2001 and 2007 Moore's Law estimates NtpmC pretty accurately. In this period most publications were on x86 platforms running Microsoft SQL Server. Starting in 2008 Moore's Law over estimated NtpmC. At the same time the number of Microsoft SQL Server publications started decreasing. 2008 saw 21 benchmark publications, 2009 saw 6 benchmark publications and, so far, 2010[3] has seen 5 benchmark publications. This is mostly because of Microsoft's increased interest in the new TPC-E benchmark, introduced in 2007, while Oracle's DBMS took the lead in a number of TPC-C publications. Also, during this period the number of cores per processor for x86 processors increased to six. Another reason for the decline in the number of TPC-C publication is the increased benchmark implementation cost, mainly due to an increased number of spindles required to cope with increase in processor speed.

Ignoring the two time periods during which NtpmC differed slightly from Moore's Law, TPC-C performance increases are remarkably similar to the processor performance improvements. This suggests that TPC-C performance is attributed solely to

[3] As of June, 2010.

processor improvements, i.e. other system components, including system software, have not contributed much to the overall performance improvement of TPC-C results. However, this does not necessarily indicate that other system components and system software haven't improved over time – quite to the contrary. The ongoing improvements of processor speeds cause challenges on other system components, such that they do not become the next system's bottleneck. Hence, their performance also needed to be increased constantly. Components whose performance improvements were slower than those of processors were replicated. For instance, the number of disks required per processor increased from a dozen in 1993 to over 100 in 2010.

In case of software such replication is not necessarily possible. Especially the DBMS of the 3rd tier cannot be replicated because of the locking nature of the benchmark application. An attempt was made to use federated databases. However, the idea was abandoned shortly after it was introduced[4]. The development of multi-core processors imposed challenges on the operating systems as well as the DBMS because of the increased number of processes that need to be scheduled to occupy all processor cores. Similarly, the increased number of processes challenged shared memory access and semaphore handling mechanisms. Similarly, the dramatic increase in memory density challenged DBMS because the design of algorithms that were previously optimized for using disk I/O, such as sort operations, hash-joins, needed to be revisited to assure that they work optimally with large amounts of memory. The dramatic increase in processor cores and the resulting increase in TPC-C database size/number of database users (see Section 2) challenged the scalability capabilities of DBMS. The fact that TPC-C applications still scale with the vast increase in processor performance is an indication that DBMS scaling capabilities improved at rate of processors performance improvements. With every DBMS release its code path increases due to the introduction of additional features. These features are not necessarily introduced to improve TPC-C, but improve the overall usability of the product, such as security and manageability features.

3.2 Comparison of TPC-C Price-Performance with Moore's Law

In this section we compare TPC-C's price-performance metric with the adjusted[5] Moore's Law. TPC-C's price-performance metric is defined as the ratio of the total system price of the SUT divided by the transactions per minute [$/NtpmC].

The triangle markers in Figure 4 shows the price per NtpmC per year from 1993 to 2010 on a logarithmic scale (base=10). The dotted line shows Moore's law as amended by David House, i.e. performance doubles every 18 months. Similarly to the previous section, as a starting point we use the price per NtpmC in 1995 as the number of results available for 1993 and 1994 is too small to indicate representative performance for those years. We calculate the Moore's Law graph with the following function:

$$f(y) = f(y_0) * 2^{-\frac{2}{3}*(y-y_0)} \quad (2)$$

[4] Reasons for the abandonment of this technology would be pure speculation.
[5] Adjusted as by David House: processor performance doubles every 18 months.

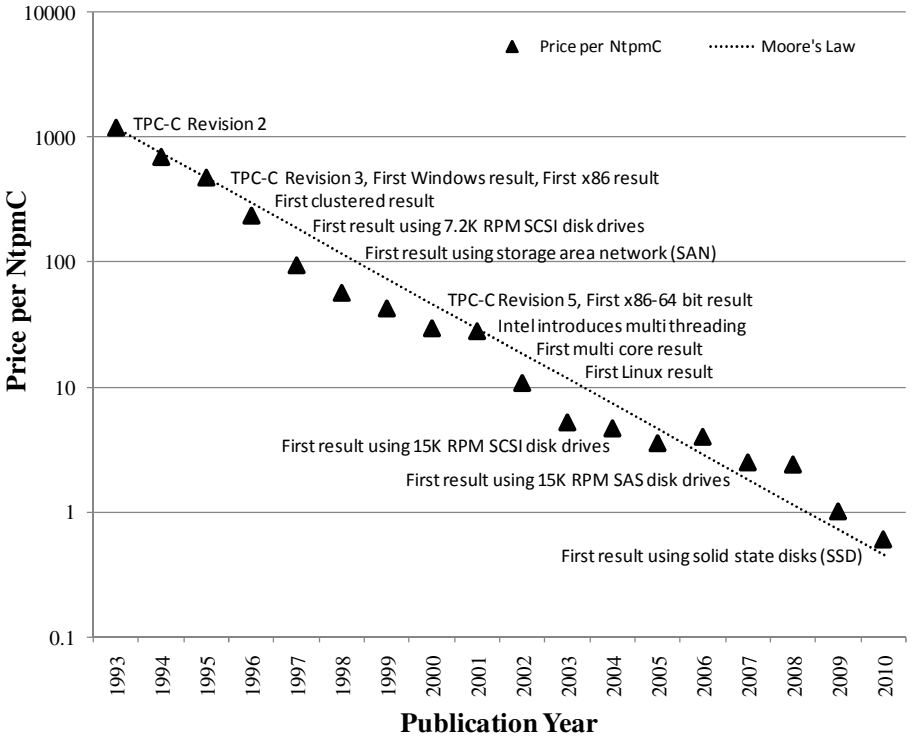

Fig. 4. TPC-C price-performance vs. Moore's Law

$f(y)$ is the price-performance projection for year y and $f(y_0) = 465.41 \frac{\$}{Ntpmc}$ is the base price-performance with $y_0 = 1995$.

The trend in Figure 4 demonstrates that the gradual decline in the cost for transaction performance in line with Moore's Law. Interestingly, we saw in the comparison of TPC-C's normalized performance metric NtpmC with the calculated Moore's Law graph, we also see a phenomenal similarity between the price-performance trend and the price-performance predicted by Moore's Law.

There are three areas where the price-performance graph slightly over and underperforms the predictions of Moore's Law. Between the years 1996 and 2000 and between 2002 and 2005 Moore's Law slightly over-calculates the average transaction price as reported by benchmark sponsors. Contrary, between 2006 and 2010 Moore's Law slightly under-calculates the actual transaction cost. However, the overall trend of the actual graph and the calculated graph are phenomenally similar.

We had a closer look at the variations but the reasons were inconclusive, except that since 2006 the price-performance have been above the expectations potentially due to the extreme drop in system component pricing caused by competition, business efficiency and factory efficiency. This trend is likely to continue for the next several years.

4 Conclusion

Many research papers and technical articles have been published about Moore's Law and its applicability to hardware performance. This paper is a first of its kind attempt to analyze Moore's theorizations in respect to the performance and the cost for performance of complex application systems like transaction processing systems. The analysis involves not just the processor perspective, but the total system perspective, including hardware and software of database servers, storage subsystems, middle tiers and component connectivity. Our analysis shows that performance and price-performance trends of over two decades of TPC-C results exhibit close resemblance to Moore's predictions.

Acknowledgements

The authors would like to thank the past and present members of the TPC committee for their commitment in producing and maintaining the TPC-C benchmark specification, which continues to be the most popular yardstick for comparing transaction processing performance and cost for performance. A special thank is given to Charles Levine for providing the historical data used in the paper. The authors would also like to thank Satinder Sethi and Ray Glasstone for their support in writing this paper, and Brean Campbell, Tony Harvey, Ven Immani and Venkatesh Kamat for their valuable comments and feedback.

References

1. Moore, G.E.: Cramming more components onto Integrated Circuits. Electronics Magazine (April 19, 1965)
2. Moore, G.E.: Progress in digital integrated electronics. In: International Electron Devices Meeting, pp. 11–13 (1975)

TPC-V: A Benchmark for Evaluating the Performance of Database Applications in Virtual Environments

Priya Sethuraman and H. Reza Taheri

VMware, Inc., Palo Alto, Ca, USA
{psethuraman,rtaheri}@vmware.com

Abstract. For two decades, TPC benchmarks have been the gold standards for evaluating the performance of database servers. An area that TPC benchmarks had not addressed until now was virtualization. Virtualization is now a major technology in use in data centers, and is the number one technology on Gartner Group's Top Technologies List. In 2009, the TPC formed a Working Group to develop a benchmark specifically intended for virtual environments that run database applications. We will describe the characteristics of this benchmark, and provide a status update on its development.

Keywords: Transaction processing performance; virtualization; virtual machines; server consolidation; benchmarks.

1 Introduction

A few years ago, virtualization on x86 PCs was a clever way for enthusiasts to run multiple Windows and Linux operating environments simultaneously on the same PC platform. Today, virtualization is an essential datacenter technology, enabling capabilities that would not have been possible otherwise. This has brought about a demand for virtualization benchmarks. In this paper, we will introduce virtualization and its benefits, discuss existing benchmarks, and describe the TPC-V benchmark.

It is important to point out that TPC-V is entering its early development stages. We will attempt to describe the benchmark in full, but some of those descriptions might change drastically during the development process. The intent is not to pre-announce a benchmark specification, or to promise exact properties for the benchmark. This is a status report of where the benchmark development stands at this point in time.

2 State of Virtualization

A virtual machine (VM) is a software computer that, like a physical computer, runs an operating system and applications. An operating system installed on a virtual machine is called a *guest* operating system. Virtual machines run on host servers. The same server can run many virtual machines. Every VM runs in an isolated environment. So one VM may be running a Windows-based web server application while another runs a Linux-based database application, all on the same hardware platform. The flexibility

R. Nambiar and M. Poess (Eds.): TPCTC 2010, LNCS 6417, pp. 121–135, 2011.

and many benefits that this architecture affords the user have catapulted virtualization to the top of Gartner Group's Top Technologies List [7].

In the 1960s, IBM's VM operating system permitted the execution of a variety of IBM operating systems in multiple VMs on a single hardware platform [3]. Virtualization on the Intel x86 architecture was introduced in the 1990s [6, 11, 14] as a means of allowing multiple operating systems to run simultaneously on a single PC. Today's enterprise-class virtual machine management system (*hypervisor*) vendors include VMware, Microsoft, Citrix, KVM, and Oracle on x86, plus virtualization products on the Power [10], SPARC [13], and Intel Itanium [7, 9] architectures.

2.1 Benefits of Virtualized Servers

Consolidation: The first reason many users headed to virtualized servers was consolidation. The vast majority of datacenter servers are grossly underutilized. In this scenario, several underutilized servers are converted into VMs that all run on the same host server. With today's multi-core, multi-socket commodity servers, the CapEx and OpEx savings of virtualization are major driving forces in the data center.

Resource Management: packing multiple VMs onto a single server allows the hypervisor to allocate resources to each VM based on its demand. Consolidation also enables elasticity: A VM gets more resources when its load risess. A server can achieve greater efficiency by overlapping the peaks and valleys of its many VMs.

Migration: The ability to migrate a VM [12] to a new physical server while the applications on the VM continue to be in use allows for a rich set of load balancing and resource management features. Virtualization is the fundamental enabling technology behind cloud computing.

Maintainability: The ability to migrate VMs live between hosts frees the original server for maintenance operations

HA: Achieving high availability (HA) by allowing a VM to be restarted on a new server if the server running the VM fails [20]. Virtualization-enabled HA allows a few generic servers to act as the backup for a large number of active servers because the properties of the operating environment are captured in the VM.

FT: Fault-tolerance on generic servers without hardware fault-tolerance features [21]. Two VMs are run in lockstep as is done in traditional hardware fault-tolerant architectures. The Virtual Machine Monitor (VMM) ensures externally-visible output, e.g., network packets, are sent out from only one VM, but all VMs receive copies of incoming external stimuli, e.g., reads from disk.

3 Virtualization Benchmarks

The interest in virtualized servers has meant a strong interest in performance benchmarks to study, compare, and tune virtual environments. Several benchmarks are in various states of use in the industry. The two main benchmarks are:

- SPECvirt_sc2010 [16] is an industry-standard benchmark developed by the Standard Performance Evaluation Corporation (SPEC) and released in

July 2010. It modifies a number of SPEC workloads such as SPECWeb, SPECjAppServer and SPECmail for measuring virtualized server consolidation. SPECvirt_sc2010 is expected to find widespread use.

- VMmark [22] was developed by VMware, primarily for its vSphere hypervisor operating system. But it should be possible to run VMmark on other hypervisors. VMmark is the de facto industry standard benchmark at the moment, with more than 91 published results from 13 vendors.

3.1 The Need for a Database-Centric Virtualization Benchmark

Virtualized database applications have been running under the IBM VM operating system for decades. Although skepticism existed in the early days of x86 virtualization as to whether a virtual machine could handle the demanding load of a database server, customers now routinely virtualize their OLTP and DSS servers. Virtualization at the hypervisor level provides excellent abstraction because each DBA has a hardened, isolated, managed sandbox. There is strong isolation between the VMs, which allows each VM/database to maintain its own policies in security, performance and resource management, configuration management and fault isolation. Recent results [2] prove virtualized databases work very well. End users no longer have a reluctance to virtualize databases, and there is a strong demand from the user community for a virtual database benchmark.

Database workloads include many operations that can have high overheads on virtual servers: disk I/O, inter-process communication, process/thread context switching, large memory footprints and pressure on the TLB, guest OS system calls, etc. A non-database benchmark cannot do justice to a hypervisor that does well (or poorly) running database applications. None of the existing virtualization benchmarks has a database-centric workload. Hence the strong need for a new benchmark that stresses virtualized servers with a database-centric workload.

4 Genesis of the TPC-V Benchmark

We presented a paper at the 2009 TPCTC [2], advocating the need for a benchmark to measure the performance of virtual environments under a database-centric workload. The TPC followed up on this by forming a Working Group of 14 companies[1] to scope a virtualization benchmark. To meet the charter that the TPC General Council gave the Working Group, TPC-V has to meet these requirements:

- Satisfies the industry need for a benchmark that:
  - Has a database-centric workload
  - Stresses the virtualization layer
  - Has a moderate number of VMs, exercising enterprise applications
  - Has a healthy storage and networking I/O content; emphasizes I/O in a virtualized environment
  - Does not contain many non-database application environments in an application consolidation scenario

[1] The 14 companies were: AMD, Cisco, Dell, HP, IBM, Intel, Microsoft, NEC, Oracle, ParAccel, Sun, Sybase, Unisys, and VMware.

- TPC-V results will not be comparable to other TPC benchmarks
- TPC-V generates information not covered by other benchmarks
- The benchmark has to have a timely development cycle (1-2 years) to satisfy the demand that member companies get from their customers
    - The TPC-V benchmark will be based on the TPC-E benchmark and borrows a lot from it
    - But is a different workload mix and the results cannot possibly be compared to TPC-E results.

5 TPC-V Properties

It is worthwhile to again remind the reader that this is a status report on the state of the benchmark as it stands at the publication date of this paper. The Development Subcommittee is charged with creating the final specification. Much can change after the results of prototyping experiments have been turned into the subcommittee.

5.1 Business and Application Environment

A typical business requires multiple applications to manage a variety of operations. Often these applications have been located on separate systems. With advances in virtualization technologies and in the strength of computing resources, it is now possible to co-locate these applications on the same system.

While it may be possible to install and use multiple applications in a single system image, there can be advantages to maintaining the applications in distinct virtual machines (VMs). Depending on the size of the business and the size of the system used, the business model of TPC-V may be viewed as a "Data Center in a Box", with a wide variety of applications, including both database tiers and application-management tiers all residing on logically distinct VMs within a single computer

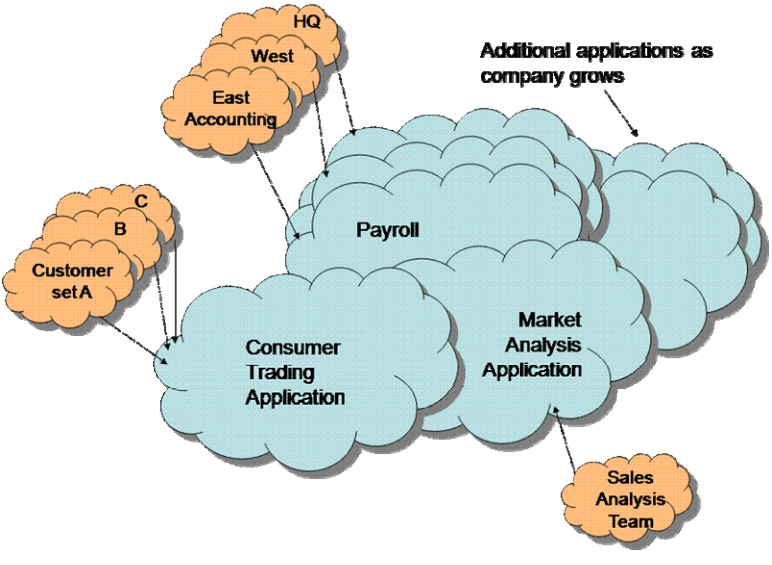

Fig. 1. Business Model: Data Center in a Box

system. The following diagram illustrates the potential complexity of the business model portrayed in the benchmark.

However, the complexities of the modeled environment do not lend themselves to a measureable, repeatable benchmark. So, the TPC-V application in Fig. 2 is a simplified view of this complex environment: retaining the key features of the business model, while effecting meaningful and comparable benchmark results.

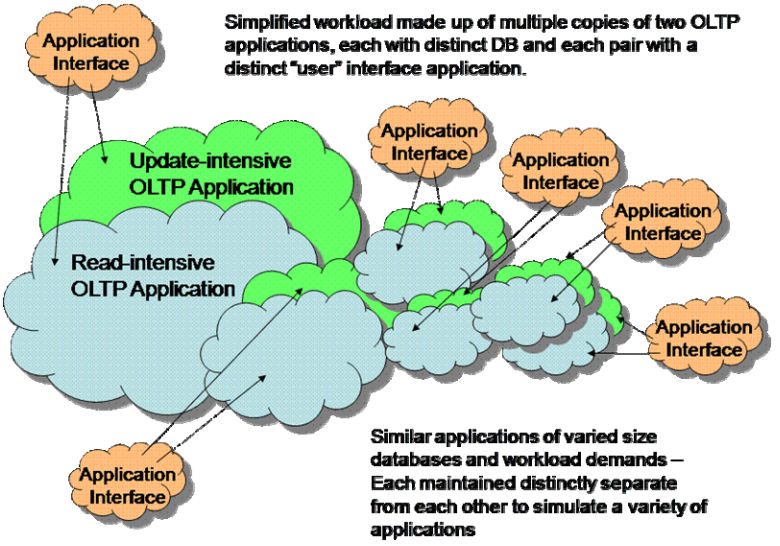

Fig. 2. Simplified VM Components

Each group of Application Interface, Update-Intensive and Read-Intensive VMs is a distinct "Set". The size and number of sets is determined by the relative total load on the system. Sets are logically distinct from each other from an application perspective, although the benchmark driver may coordinate the amount of work being required of each set.

To provide a meaningful application environment with database components and transactions that are relevant and understandable, the application environment of TPC-E is employed. TPC-E is altered to provide the desired read-intensive and update-intensive environments in Fig. 2. While TPC-E uses a business model of a brokerage house with transactions driven from multiple sources, the deployment of the adjusted application in TPC-V is intended to represent a wider variety of OLTP-based applications that could be employed in a virtualized computing environment.

5.2 The TPC-E Benchmark

TPC Benchmark™ E [17], TPC-E for short, is TPC's most up to date transaction processing benchmark. It models the activity of a brokerage firm that must manage customer accounts, execute customer trade orders, and be responsible for the interactions of customers with financial markets. TPC-E transactions are executed against

three sets of database tables that represent market data, customer data, and broker data. A fourth set of tables contains generic dimension data such as zip codes. The following diagram illustrates the key components of the environment:

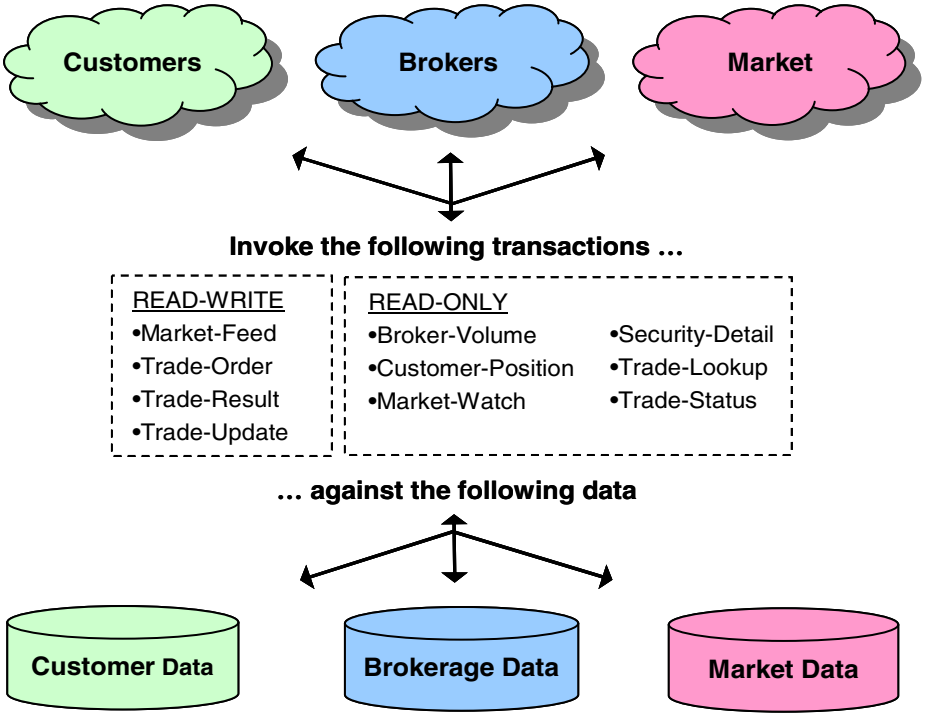

Fig. 3. TPC-E Application Components

The benchmark represents a simplified form of the application environment. To measure the performance of the OLTP system, a simple Driver generates Transactions and their inputs, submits them to the System Under Test (SUT), and measures the rate of completed Transactions being returned. To simplify the benchmark and focus on the core transactional performance, all application functions related to User-Interface and Display-Functions have been excluded from the benchmark. The SUT is focused on portraying the components found on the sever side of a transaction monitor or application server.

5.3 TPC-V Design Considerations

5.3.1 TPC-V Schemas and Tables

TPC-E defines 33 tables and 10 transactions types. TPC-V will retain the basic schema and transaction types of TPC-E. But some properties will have to be modified to facilitate the creation and loading of many different database sizes in one SUT, and to route different transactions to different VMs (see section 5.3.4).

A TPC-E SUT is divided into two Tiers:

- Tier A contains all hardware and software needed to implement the down-stream Connector, EGenTxnHarness**,** Frame Implementation, and Database Interface functional components. Frame Implementation is a sponsor provided functionality that accepts inputs from, and provides outputs to, EGenTxnHarness through a TPC Defined Interface. The Frame Implementation and all down-stream functional components are responsible for providing the appropriate functionality outlined in the Transaction Profiles. EGenTxnHarness defines a set of interfaces that are used to control the execution of, and communication of inputs and outputs, of Transactions and Frames.
- Tier B contains all software needed to implement the Database Server functional components.

5.3.2 TPC-V Set of VMs

Using the TPC-E transactions as a base, the working group has defined 3 VMs that together form a *Set* for the TPC-V benchmark. The functionality of the Tier B component of the TPC-E SUT has been divided into two separate VMs. One VM handles the Trade-Lookup and Trade-Update transactions, simulating the high storage I/O load of a decision support environment. The second VM services all other transactions, which have a CPU-heavy profile and represent an OLTP environment. It remains to prototyping to ensure that the two VMs display the desired characteristics.

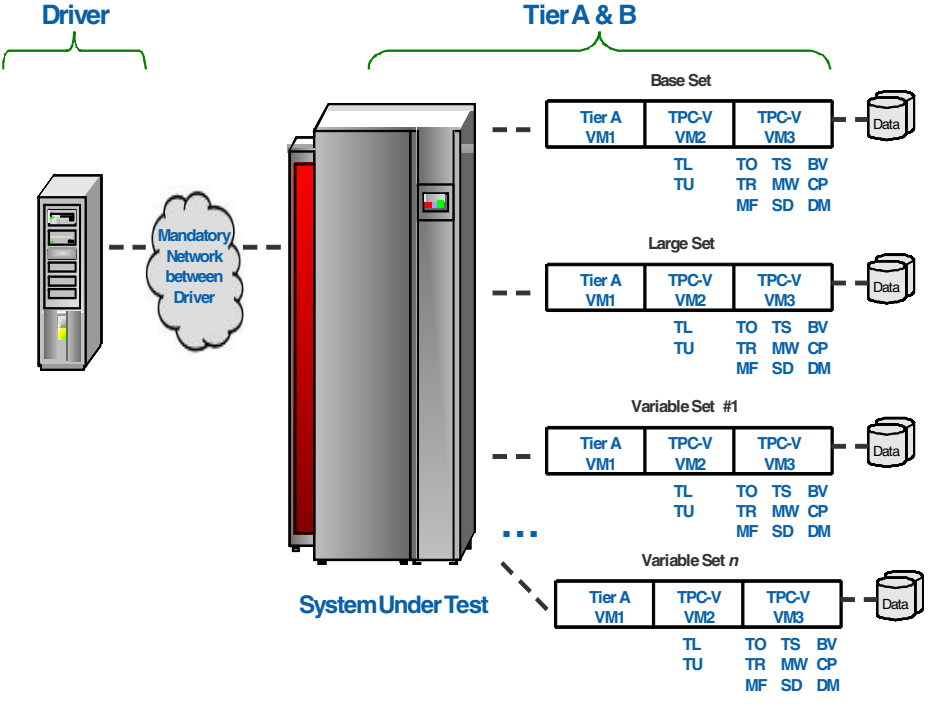

Fig. 4. Sample Components of Physical Test Configuration

Tier A in TPC-V functions similarly to a TPC-E Tier A with one major difference: Based on the transaction type, it routes the transaction to one of the two Tier B VMs. In Fig. 4, notations TL, TU, etc. under the VMs are the 2-letter abbreviations of TPC-E transactions [16].

In Fig. 5, Customer Emulator (CE) is responsible for emulating customers, requesting a service of the brokerage house, providing the necessary input for the requested service, etc. Market Exchange Emulator (MEE) is responsible for emulating the stock exchanges: providing services to the brokerage house, performing requested trades, providing market activity updates, etc. Rather than describing in detail the functions of each component, we point out that the benchmark has a large number of components defined in the TPC-E specification. Some components, in yellow, have to be commercially available products. Some, in turquoise, are provided by the TPC and have to be incorporated in the benchmarking harness as is. The rest, in purple, is left up to the sponsor to provide.

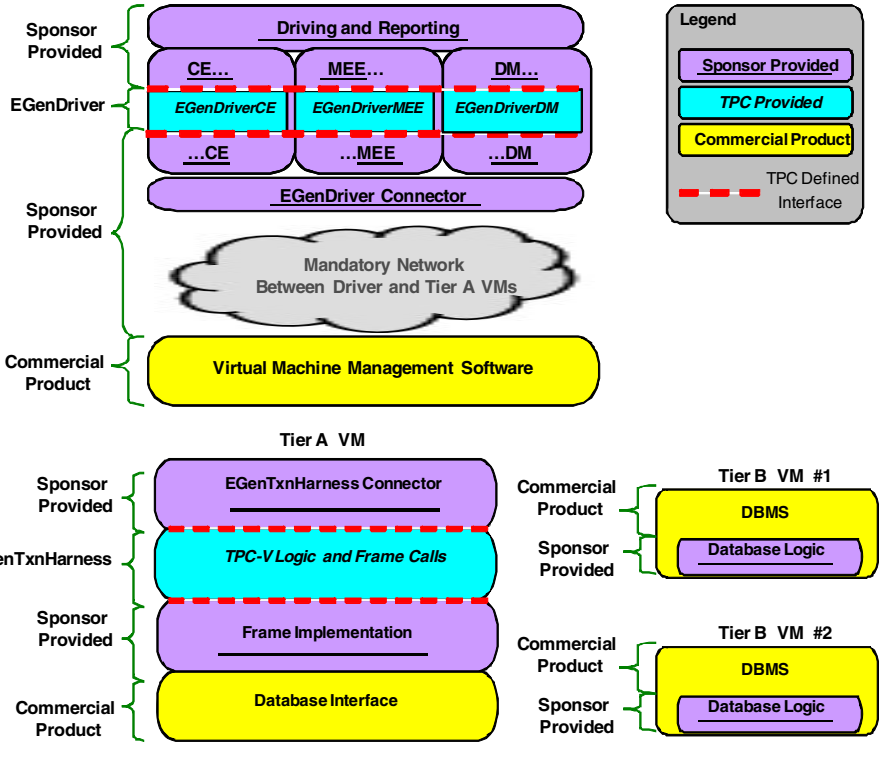

Fig. 5. Functional Components of the Test Configuration

5.3.3 Core Decisions Regarding Benchmark Load

One of the first decisions the working group made was to depart from the *tile* architecture used in other virtualization benchmarks [3, 22]. In those benchmarks, a tile was defined to have a certain load that was the same for all tested systems. Two of the

most desirable properties of TPC benchmarks are their applicability to a wide range of server sizes, and their long lifetimes. TPC benchmarks scale, and they last. But a constant load per tile would mean the number of tiles would increase with the performance of the system, possibly to unreasonably large numbers. In practice, more powerful servers support more VMs; but they also host VMs with higher resource demands. For example, it would be reasonable for a virtualization benchmark that runs on a 256-core server to have a VM that consumes 16 cores' worth of resources. But unless the benchmark scales down the demand to each VM as the power of the server goes down, that same benchmark cannot be run on an 8-core server. A benchmark that is intended to be applicable to a wide range of servers, as is the tradition of TPC benchmarks, especially one that is meant to stand the test of time, has to deal with this issue. So early on, the working group decided against a tile architecture with a constant load per tile.

Another common practice in existing virtualization benchmarks is to require a certain number of virtual processors, or a certain memory size, for each VM [3, 4, 22]. TPC-V does not specify such configuration details. If each VM is required to have, for example, 2 virtual processors, then a host with 1024 physical processor cores would need at least 512 VMs to saturate the system assuming each VM fully utilized its virtual processors. 512 VMs might be an unreasonably large number of VMs. Furthermore, what will happen in 10 years, when there might be servers with an order of magnitude more cores? So, hard-coding the number of virtual processors, or the memory size, of the VM would make it hard for the benchmark to be meaningful across a range of server sizes, or have a long lifetime. TPC benchmarks have always been defined in terms of the load presented to the system (mix of transactions, table properties, etc.), not the characteristics of the system (a particular memory size, a specified number of physical or virtual processors, etc.).

If we move away from a tile architecture with a fixed load, how should the load be defined? A first, intuitive approach is to base the load on the *size* of the system, and what better way to gauge the size of a system than its number of processors/cores?

That approach might work when dealing with a single architecture. But not all processor cores are comparable across a range of architectures: x86, Power, SPARC, UltraSPARC T3 [14], Intel Itanium, etc. If this benchmark is to be a suitable choice for measuring the performance of virtualized servers irrespective of the processor architecture, one cannot base the definition of the benchmark on the number of cores or the memory size of the physical server.

5.3.4 The Set Architecture

5.3.4.1 Basing the Load on the Performance of the Server. To avoid the limitations detailed in the prior sections, the working group has devised a **Set** architecture whereby both the number of Sets, and the load placed on each Set grow as the performance of the system increases. The advantage here is that the benchmark will emulate the behavior of real-world servers: More powerful servers host more VMs, but also VMs that handle more load. This ensures that TPC-V is a fitting benchmark for servers of all sizes, and it will stay relevant in the future as servers become more powerful. It will scale, and it will last.

5.3.4.2 Varying the load across Sets. A shortcoming of existing benchmarks is that the same exact load is placed on all tiles (or VMs). In the real world, a virtualized host has to deal with the challenge of managing resources across VMs with varying demands. So each Set in a TPC-V configuration will contribute a different percentage of the overall throughput.

The exact number of Sets and the percentage contributed by each Set will depend on the prototyping experiments in the coming year. Let us assume that the metric for TPC-V is in terms of transactions per second, and let us abbreviate that to tpsV (the exact benchmark metric is yet to be named and defined). With that, following are the numerical values that will be used to initiate the prototyping process:

- A Base Set, which contributes 15% of the overall throughput of the SUT
- A Large Set, which contributes 45% of the overall throughput of the SUT
- Variable Sets contribute the remaining 40% of the overall throughput. The exact number of Variable Sets and the division of the 40% among them is calculated based on the performance of the SUT. More powerful systems will have more Variable Sets, determined by:

$$f(tpsV) = \max(1, SQRT(45\% * tpsV) / M + C)$$

where M=SQRT(40) and C=-2.

The actual number of Variable Sets will have some flexibility in order to avoid the case where a slight change in performance can force the benchmark sponsor to reconfigure the SUT for a different number of Sets.

The table below is an example of the Set contributions to the overall throughput:

Table 1. TPC-V SET/VM/VP Sizing Worksheet

SUT Target tpsV=>	1,000	2,000	4,000	8,000	16,000	32,000
Base Set Ratio	15%	15%	15%	15%	15%	15%
Large Set Ratio	45%	45%	45%	45%	45%	45%
Variable Sets Ratio	40%	40%	40%	40%	40%	40%
f(tpsV)	1.35	2.74	4.71	7.49	1.42	16.97
M	6.32	6.32	6.32	6.32	6.32	6.32
C	2.00	2.00	2.00	2.00	2.00	2.00
variability (+/- N)	0.50	0.50	0.50	0.50	0.50	0.50
min Variable Sets	-	2.0	4.0	6.0	10.0	16.0
max Variable Sets	1.0	3.0	5.0	7.0	11.0	17.0
Tier A VMs per Set	1	1	1	1	1	1
Tier B VMs per Set	2	2	2	2	2	2
Base Set tpsV	150	300	600	1,200	2,400	4,800
Large Set tpsV	450	900	1,800	3,600	7,200	14,400
Variable Set tpsV	400	800	1,600	3,200	6,400	12,800
Total Sets (S+L+M)	3	5	7	9	13	19
Small Set VMs	3	3	3	3	3	3
Large Set VMs	3	3	3	3	3	3

Table 1. *(continued)*

Medium Set VMs	3	9	15	21	33	51
Max Total VMs	9	15	21	27	39	57
Variable Set 1 tpsV	400	133	107	114	97	84
Variable Set 2 tpsV	-	267	213	229	194	167
Variable Set 3 tpsV	-	400	320	343	291	251
Variable Set 4 tpsV	-	-	427	457	388	335
Variable Set 5 tpsV	-	-	533	571	485	418
Variable Set 6 tpsV	-	-	-	686	582	502
Variable Set 7 tpsV	-	-	-	800	679	586
Variable Set 8 tpsV	-	-	-	-	776	669
Variable Set 9 tpsV	-	-	-	-	873	753
Variable Set 10 tpsV	-	-	-	-	970	837
Variable Set 11 tpsV	-	-	-	-	1,067	920
Variable Set 12 tpsV	-	-	-	-	-	1,004
Variable Set 13 tpsV	-	-	-	-	-	1,088
Variable Set 14 tpsV	-	-	-	-	-	1,171
Variable Set 15 tpsV	-	-	-	-	-	1,255
Variable Set 16 tpsV	-	-	-	-	-	1,339
Variable Set 17 tpsV	-	-	-	-	-	1,422
Total Variable Set tpsV	400	800	1,600	3,200	6,400	12,800

5.3.5 Elasticity

Performance benchmarks are typically measured in "steady state", where the flow of work requests is adjusted to meet the capabilities of the system. For a single application, this can provide a satisfactory answer, but not for a virtualized server. The following diagram illustrates the existence of workload dynamics in the business model for TPC-V. Each application may vary between the minimum and maximum requirements, depending on such things as time zone, time of day, time of year or introduction

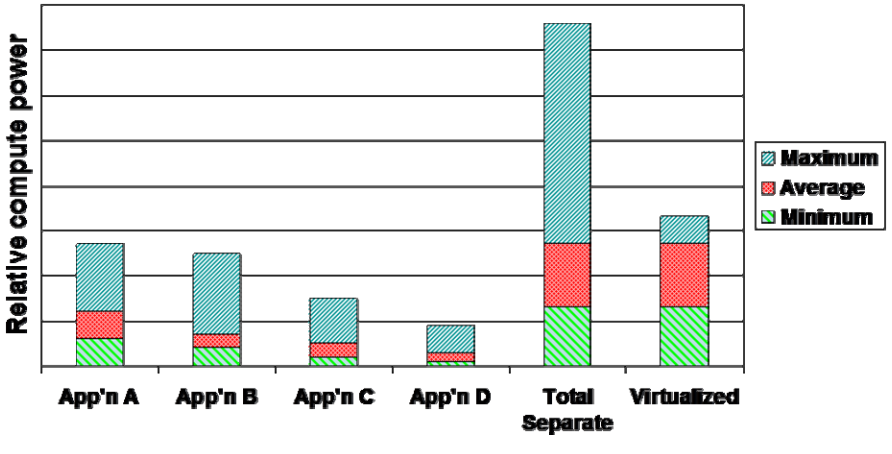

Fig. 6. Demands by workload

of a new product. To accommodate each of the four applications represented on separate systems, the total compute power required is represented by the "Total Separate" bar. However, in the chosen business model, the peak workload demands for each application are not simultaneous. One workload may be at a peak when another is at a valley, allowing computer resources to be shifted from the low-use application to the high-use one for some period of time, and shifting the resources to another high-demand application at a subsequent point. This allows the total configured capacity to be more like the bar marked "Virtualized."

In the environment modeled by the benchmark, the dynamic nature of each workload could be dictated by a wide variety of influences that result in an unpredictable shifting of resources and an equally unpredictable amount of overall system output. As with the complexity of the modeled application environment, this level of workload dynamics is not easily repeated to deliver comparable measurements. Since the primary requirement of the virtualized environment for this situation is the ability to dynamically allocate resources to the VMs that are in high demand, it is sufficient to define a workflow time line that shifts workload demands among the VMs in a predictable manner, as illustrated, below:

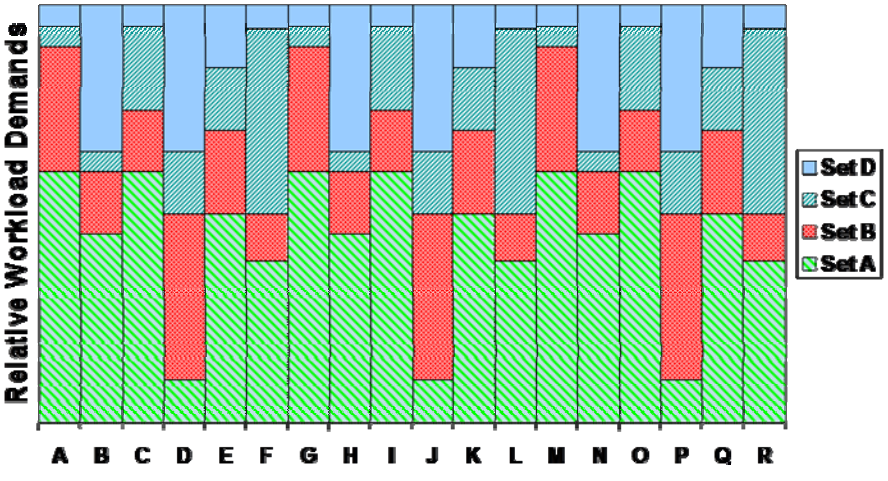

Fig. 7. Time Periods

If we consider 4 applications represented by 4 sets (A-D), then over time, each set's load vaires, and hence the contribution to the overall system throughput varies.

Using the numerical values in the draft specification, below are the actual variations that we envision for the benchmark. There will be 12 10-minute periods in the 2-hour Measurement Interval, during which the load of each Set will vary by as much as a factor of 2, 3, or 4 for the Large, Base, and Variable Sets, respectively. The exact values will be determined after the prototyping period is completed. Applying the values in Table 2 to Figure 8, Set A would be the large set, set D the base set, B and C the variable sets.

Table 2. Variation of the load of TPC-V Sets

Period	Large Set	Base Set	Variable Sets
1	45%	15%	40%
2	40%	10%	50%
3	45%	10%	45%
4	55%	30%	15%
5	35%	10%	55%
6	50%	15%	35%
7	30%	10%	60%
8	45%	15%	40%
9	50%	15%	35%
10	60%	20%	20%
11	40%	15%	45%
12	45%	15%	40%
Average	45.0%	15.0%	40.0%

6 Benchmark Development Status

The Working Group has produced a specification draft that was accepted by the TPC General Council at its June, 2010 meeting. A Development Subcommittee, with 11 member companies, has been formed to continue the development of the benchmark. Our goal is to complete the process by the end of 2011, and meet the original goal of a 1- to 2-year development period.

6.1 Challenges

Although the Working Group has been able to find common ground among the member companies in producing a draft specification, and in resolving all issues that have been raised over the past 6 months, we do recognize that developing a complex benchmark with a short schedule is a great challenge.

6.1.1 Which Virtualization Properties to Measure

In section 2.1, we listed 6 benefits for virtualized servers. In an ideal world, the TPC-V score should be dependent on a virtual system performing well in all the areas listed in 1.1, and perhaps many more similar ones. Given real world constraints, TPC-V will measure how well a virtualized server does in consolidation and resource management. If a hypervisor is able to meet the TPC-V requirements on multiple server nodes, then the ability to migrate VMs live between hosts will also be highlighted by TPC-V. Although high availability and fault tolerance are interesting new research areas for the TPC [19], the working group will not be able to address those properties given the development schedule of the first revision of TPC-V.

6.1.2 Development Schedule

TPC benchmarks often take 3 to 5 years to develop. But a virtualization benchmark is in demand today, and the council does not have the luxury of 3 to 5 years. So the

working group is starting from the solid foundation of TPC-E, and limiting the new benchmark properties to what can be tested and turned into a specification in 2 years.

6.1.3 Architecture-Agnostic Definitions

Many existing virtualization benchmarks are defined specifically for the x86 architecture [3, 4, 22], and even specifically for one hypervisor [22]. In such cases, it is easy to assume what a VM is, what a hypervisor is, what a virtual processor is, what a core is, etc. The strength of TPC benchmarks is applicability across a wide variety of architectures and server sizes. The working group cannot define a benchmark that assumes a certain hardware or software architecture.

To deal with this challenge, the TPC-V workload does not require, e.g., a certain number of virtual processors, a certain guest operating system, or any particular VM properties. We believe that the working group has succeeded in creating a benchmark that merely defines the workload. The SUT properties needed to deal with the workload follow from the workload definition.

For example, virtualized servers typically have a degree of *oversubscription* (also called overcommitment). In an oversubscribed system, each VM is configured with enough virtual processors to handle its peak load. But most of the time, the VM uses a fraction of its possible peak CPU utilization. Thus, the total number of virtual processors configured by all the VMs is higher (often much higher) than the number of physical processor cores or threads on the host server. But since not all VMs run at peak load simultaneously, the server is not overtaxed. Rather than emulating this explicitly by requiring a specific degree of oversubscription, or by specifying the number of virtual processors in the VMs, the benchmark uses the elasticity properties described in section 5.3.5 to emulate oversubscription without actually spelling it out in the specification.

Acknowledgements

The TPC-V benchmark is being developed through the efforts of contributors from more than a dozen companies. Much of the material reported here is the result of the work of the committee members as a whole, rather than the authors. We would in particular like to acknowledge the contributions of John Fowler, Karl Huppler, Doug Johnson, Matthew Lanken, Jody McCoy, Arno Mihm, Pete Peterson, Francois Raab, Cecil Reames, Jamie Reding, and Wayne Smith.

References

1. Agesen, O.: Software and Hardware Techniques for x86 Virtualization (2009), http://www.vmware.com/files/pdf/software_hardware_tech_x86_virt.pdf
2. Bose, S., Mishra, P., Sethuraman, P., Taheri, R.: Benchmarking Database Performance in a Virtual Environment. In: Nambiar, R.O., Poess, M. (eds.) TPCTC 2009. LNCS, vol. 5895, pp. 167–182. Springer, Heidelberg (2009)
3. Casazza, J., Greenfield, M., Shi, K.: Redefining Server Performance Characterization for Virtualization Benchmarking. Intel® Technology Journal (August 2006), http://www.intel.com/technology/itj/2006/v10i3/

4. De Gelas, J.: Real-world virtualization benchmarking: the best server CPUs compared, http://www.anandtech.com/show/2770/1
5. Creasy, R.J.: The Origin of the VM/370 Time-Sharing System. IBM Journal of Research and Development 25(5), 483
6. Figueiredo, R., Dinda, P.A., Fortes, J.A.B.: Guest Editors' Introduction: Resource Virtualization Renaissance. Computer 38(5), 28–31 (2005), http://www2.computer.org/portal/web/csdl/doi/10.1109/MC.2005.159
7. Gartner Identifies the Top 10 Strategic Technologies for 2009: Gartner Newsroom (2009), http://www.gartner.com/it/page.jsp?id=777212
8. HP Integrity Virtual Machines, http://h71028.www7.hp.com/enterprise/us/en/os/hpux11i-partitioning-integrity-vm.html?jumpid=ex_r1533_us/en/large/tsg/go_integrityvms
9. HP Integrity Virtual Machine User Guide, http://bizsupport2.austin.hp.com/bc/docs/support/SupportManual/c02044090/c02044090.pdf
10. IBM PowerVM Virtualization, http://www-03.ibm.com/systems/power/software/virtualization/index.html
11. Nanda, S., Chiueh, T.-C.: A Survey on Virtualization Technologies. Technical Report ECSL-TR-179, SUNY at Stony Brook (Feburary (2005), http://www.ecsl.cs.sunysb.edu/tr/TR179.pdf
12. Nelson, M., Lim, B.-H., Hutchins, G.: Fast Transparent Migration for Virtual Machines. In: USENIX 2005, pp. 391–394 (April 2005)
13. Oracle Sun Virtualization, http://www.sun.com/software/solaris/virtualization.jsp
14. Patel, S., Phillips, S., Strong, A.: Sun's Next-Generation Multi-threaded Processor - Rainbow Falls: Sun's Next Generation CMT Processor. HOT CHIPS 21
15. Rosenblum, M., Garfinkel, T.: Virtual Machine Monitors: Current Technology and Future Trends. Computer 38(5), 39–47 (2005)
16. SPEC Virtualization Committee, http://www.spec.org/virt_sc2010
17. TPC: Detailed TPC-E Description, http://www.tpc.org/tpce/spec/TPCEDetailed.doc
18. Transaction Processing Performance Council, http://www.tpc.org
19. Vieira, M., Madeira, H.: From Performance to Dependability Benchmarking: A Mandatory Path. In: Nambiar, R., Poess, M. (eds.) TPCTC 2009. LNCS, vol. 5895, pp. 67–83. Springer, Heidelberg (2009)
20. VMware Inc., VMware High Availability, Concepts, Implementation, and Best Practices (2007), http://www.vmware.com/files/pdf/VMwareHA_twp.pdf
21. VMware Inc., Protecting Mission-Critical Workloads with VMware Fault Tolerance (2009), http://www.vmware.com/files/pdf/resources/ft_virtualization_wp.pdf
22. VMware Inc., VMmark: A Scalable Benchmark for Virtualized Systems, http://www.vmware.com/pdf/vmmark_intro.pdf

First TPC-Energy Benchmark: Lessons Learned in Practice

Erik Young, Paul Cao, and Mike Nikolaiev

Hewlett-Packard Company, 11445 Compaq Center Dr. W, Houston, TX-77070, USA
{erik.young,paul.cao,mike.nikolaiev}@hp.com

Abstract. TPC-Energy specification augments the existing TPC Benchmarks with Energy Metrics. The TPC-Energy specification is designed to help hardware buyers identify energy efficient equipment that meets both their computational and budgetary requirements. In this paper we discuss our experience is publishing industry's first-ever TPC-Energy metric publication.

Keywords: TPC, Benchmarks, Energy, Performance, HP, TPC-C, TPC-E ProLiant.

1 TPC-Energy Specification

Performance and price/performance metrics have been key criteria in data center purchasing decisions, but the demands of today's corporate IT environment also includes energy consumption as a primary consideration. Energy efficiency has become a significant factor in evaluating computing systems. To address this shift of priorities, the TPC has developed the Energy Specification which enhances its widely used benchmark standards.

The TPC-Energy specification augments existing TPC Benchmark Standards, including TPC-C, TPC-E and TPC-H, adding the measurement of energy consumption and relating this to the performance achieved. TPC-Energy enables manufacturers to provide energy metrics in the form "Watts per performance," where the performance units are particular to each TPC benchmark. As vendors publish TPC-Energy results, customers will be able to identify systems, via the TPC Web site, that meet their price, performance and energy requirements.

1.1 TPC-Energy Metrics Characteristics

Performance, cost of ownership, and energy utilization of computing systems are in some ways opposing characteristics. A measure of "goodness", of a system design can be framed as having maximum performance, with the minimum cost and the minimum energy consumption. There is also the characteristic of the how to maximize the utilization of the energy being consumed. This efficiency could be measured in a variety of ways, but in the case of the TPC-Energy benchmark metrics, efficiency is a direct correlation to the performance. Representing each of these metrics on an axis provides a simple visual indicator of this "goodness".

R. Nambiar and M. Poess (Eds.): TPCTC 2010, LNCS 6417, pp. 136–152, 2011.

As illustrated in Figure 1, the 100% value of each axis is the baseline for comparison and the shaded portion illustrates the "goodness" of reducing the energy consumption and cost without affecting the performance. This is the ideal case, but in computing systems being deployed today, some acceptable performance reduction can have a significant effect on the reductions in energy consumption and costs.

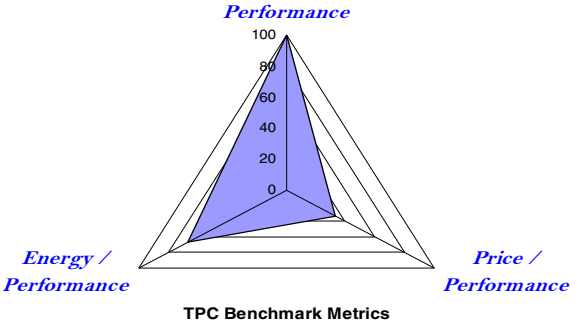

Fig. 1.

1.2 Intelligent Configuration

In addition to the vast amount of engineering design to reduce energy consumption of today's computing hardware, the intelligent selection and configuration of the available systems components is deterministic in the energy efficiency of the actual deployment of an IT application. In some application environments the maximum performance is the overriding criterion and energy efficiency and cost have a small influence on the intelligent selection of components. However, the vast majority of corporate IT deployments are measured on the return of investment (ROI). This ROI includes the operational costs in addition to the initial deployment investment.

The information disclosed herein highlights the energy efficiency considerations when deploying Hewlett Packard ProLiant systems running database applications. These same considerations and analyses can be applied to most vendors' hardware solutions today. In actual customer deployments, intelligent component selection incorporates more than just the hardware analysis, as the performance of the software selection and tuning has a direct impact on the overall system performance and can also contribute to reductions in energy consumption during periods of reduced system load. The impact analysis of software selection is not included in this paper.

2 The Benchmark Configuration

Computing systems deploying database applications can generally be defined using three major subsystems and an additional miscellaneous subsystem.

1. **Database Server Subsystem**– Components which perform the intelligent data retrieval, updates, and calculation of results, using commercially available database software.

2. **Storage Subsystem** - Provides durable media and transfer mechanism for storing the information retrieved and updated by the database server.
3. **Application Server Subsystem** – The system(s) which run the business logic and typically the user interface to the database system. (This is sometimes combined with the Database Sever Subsystem)
4. **Miscellaneous Subsystem** – Additional components which are required for the systems operations, (this is typically networking and display components.)

2.1 TPC-E Benchmark Configuration

The TPC-E benchmark database schema models a brokerage firm with customers who submit transactions related to stock trades, stock account inquiries, and stock market research. In addition, the brokerage firm interacts with financial markets to execute orders on behalf of the customers and updates relevant account information [4].

The database server HP ProLiant DL580 G7 consists of four Intel 8-core processors x7560 EX 2.27 GHz, 1 TB of main memory, two 4-port Giga-bit network controllers and 10 storage HP SA P411 controllers.

The storage controllers are connected to external 40 x D2700 enclosures to the server holding a total of 900 SAS SFF disk drives. Two disks on the internal bay were used for the operating system and database software. The remaining 4 disks are connected to an integrated SA P410i controller holding the database redo log files.

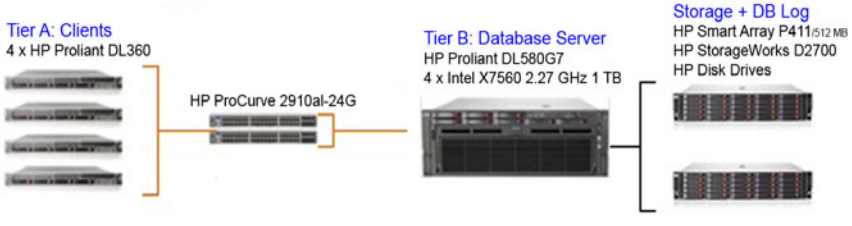

Fig. 2.

2.2 TPC-C Benchmark Configuration

The second configuration was a TPC-C benchmark conducted on the HP ProLiant DL585 G7, pictured in Figure 2. The operating system used for the benchmark was Windows Server 2008 R2 Enterprise Edition. The DBMS used was Microsoft SQL Server 2005 Enterprise x64 Edition SP3.

The storage consisted of 180 – 120GB SSD drives for the database data, two 146 GB SAS drives for the operating system, 66 - 146GB SAS drives for the database log, and 100 - 300GB drives for backup and 60 day space.

Nine LSI 92000_8E controllers were connected to nine D2700 disk drive boxes utilizing two controller ports per D2700. The HP SmartArray P410i controller was connected to the internal disk drive cage which contained the 2 - 146GB SAS drives configured as a RAID-1 logical drive. The HP SmartArray P812 was configured as RAID1+0 and connected 4 - D2700 drive boxes for backup.

A FC1242 Dual Channel 4Gb PCI-e HBA was connected to an MSA2324fc utilizing 2 HBA ports, one to each of the redundant controllers of the MSA 2324fc. The MSA2324fc cache configuration was set to fault tolerant active-active. This MSA2324fc contained 22 drives at 300GB and connected to two MSA 70 drive boxes each with 22 drives each at 146 GB for the transaction log. These were configured as 4 virtual disks at RAID 10.

The application servers consisted of 22 DL360 G5/ 1.60GHz and 2 DL360 G6 / 2.40GHz servers. The priced configuration substituted 24 DL360 G6 / 2.40GHz client systems.

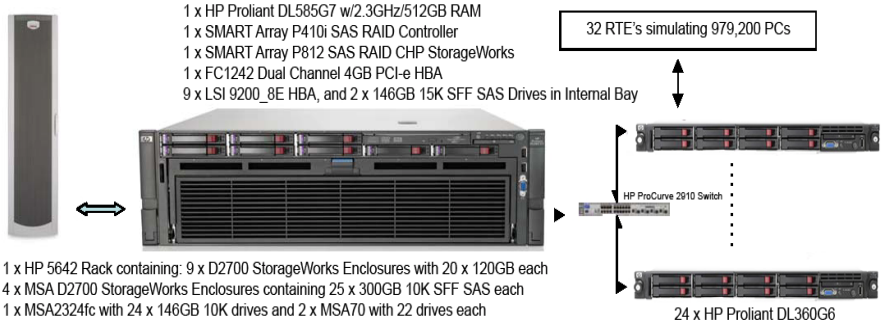

Fig. 3.

3 Energy Measurement Configuration

The energy measurement process may seem complicated and confusing for users unfamiliar with the TPC-Energy specification. This section attempts to make the connection between the specification and its real world application. It covers the individual specifications and the implementation details of the TPC-E benchmark configuration mentioned in the preceding text.

3.1 Reported Energy Configuration

The Reported Energy Configuration (REC) is based on the TPC Benchmark Standard "Priced Configuration". It consists of all the components that participate in the energy measuring process and may be divided into subsystems, each of which consists of one or more Power Measurable Units (PMU).

The TPC-E REC in Figure 4 consists of three HP ProLiant DL series servers and twenty-four disk array enclosures. These are categorized by subsystem (Application Server, Database Server and Storage) to enable the reporting of subsystem energy metrics. Not pictured is the Misc subsystem, which contains "priced components" like monitors, network switches, etc. Although, these components are required to be included in the TPC-Energy measurements, they are omitted from Figure 4 for simplicity.

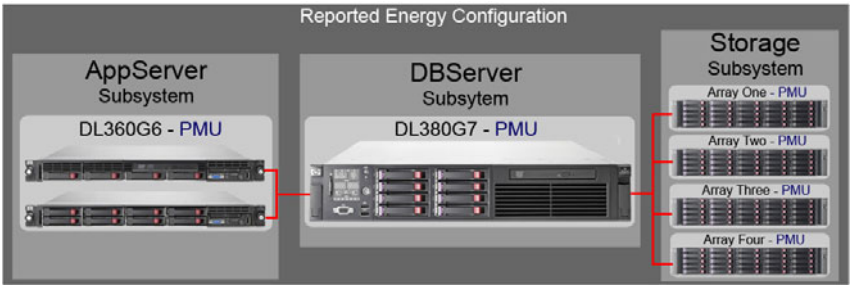

Fig. 4.

The Application Server subsystem consists of a single PMU for measuring the power consumption of two HP ProLiant DL360 G6 servers which hosts the software that provides the interface between the simulated users and the Database Server(s).

The Database Server subsystem consists of a single PMU for measuring the HP ProLiant DL380 G7 server, which hosts the database management software and implements the database and transactions of the TPC-E Benchmark.

The Storage subsystem is more complicated to measure and was divided into four PMU's, each containing eight HP StorageWorks D2700 disk array enclosures. This division was required due to the current measurement limitations of the power analyzers and the coupling selected for this particular benchmark.

3.2 Power Measurable Unit

The TPC-Energy specification, Clause 0.4, defines a Power Measurable Unit (PMU) as a component or collection of components of the REC which can be independently measured with a power analyzer. The PMU components are hardware which is instrumented for measurements using a power analyzer. The PMU energy consumption data is collected from the power analyzer via the TPC provided Energy Measurement System software as shown in

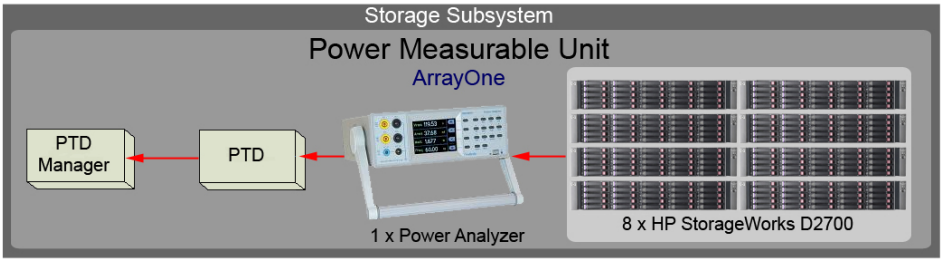

Fig. 5.

The first PMU was designated with the name ArrayOne, and is one of the four PMU's of the Storage Subsystem. These four PMUs are each a composite PMU, each containing eight HP StorageWorks D2700 disk array enclosures. Each D2700 could

be independently measured as an individual PMU. The combining of multiple units into a larger group for measurement as a single composite PMU simplifies the measurements by requiring less power analyzers.

There are of course physical limitations in the number of units which can be combined as a composite PMU. This grouping of the D2700 disk array enclosures was limited to eight per composite PMU due to the 20 ampere maximum current ratings of the power cables, breakout boxes, and power analyzers. Most electrical codes also require a safety-margin of not exceeding 80% of the maximum rating, which would be 16 amperes in this case.

3.3 PMU Substitution

The TPC-Energy specification, Clause 3.6 provides a methodology for calculating energy for PMU's which are not measured or substituted. The TPC-C Application Server subsystem of Figure 3 consisted of 24 DL360 G5 servers which are close to end-of-life, and 2 DL360 G6 servers which are the current generation of shipping servers. HP needed to price 26-DL360 G6 servers for the benchmark's priced configuration. It was therefore necessary to exercise the option in Clause 3.6.3 to measure a subset of equivalent PMUs and use those measurements to extrapolate the total energy consumption for the priced Application Server subsystem of 26 – DL360 G6 servers. The energy consumption for the unmeasured equivalent PMUs will be the extrapolated value(s) per Clause 3.6.2.

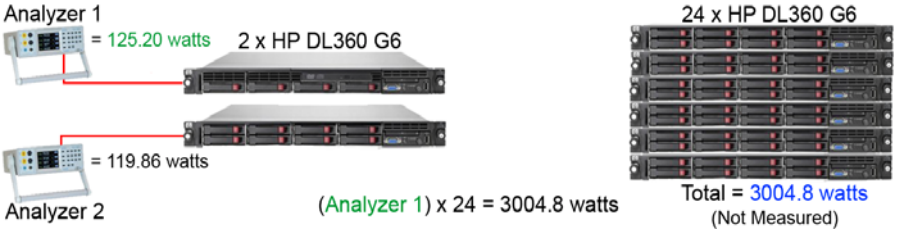

Fig. 6.

The TPC-C AppServer subsystem in Figure 6 shows the Priced configuration of 26 DL360 G6 servers. Clause 3.6.2 requires that when extrapolation is used for the energy consumption of a PMU, instead of measuring the actual energy consumption, the following requirements must be met and procedure must be followed.

1. At least two equivalent PMUs must be included in the measured configuration.
2. The Auditor selects two PMUs from the measured configuration to be measured for the extrapolation calculation. (In this case there were only two)
3. The measured PMU's energy must be less than 10% variance to be considered equivalent.
4. The energy consumption value used for each of the unmeasured PMUs is the value of the highest reading of the measured PMUs.

5. The total energy consumed for the Priced Application servers is the energy consumed by the measured PMU's plus the energy consumed by the calculated unmeasured PMUs.

Applying the above procedure, the total energy reported for the AppServer subsystem is 3249.86 watts. This is calculated by assigning the highest measurement of 125.20 watts to the 24 unmeasured severs (24 x 125.20 watts = 3004.8 watts). This extrapolated value for the substituted application servers is then added to the power of all the measured application servers.

$$\left(24 \times 125.20 watts\right) + 125.20 watts + 119.86 watts = 3249.86 watts. \quad (1)$$

3.4 Power Analyzer Configuration

The TPC-Energy specification, Clause 6 lists the requirements associated with the use of compliant and approved power analyzers. It defines the power analyzer as the device that will be connected or coupled to the power input of the System Under Test (SUT) to collect power readings during the benchmark run. More specifically, Clause 6.1.2.4 states that the analyzer must be capable of reporting watts, voltage, amperes and power factor with an average accuracy of not to exceed 2% for each set of power measurement data, for the ranges measured during the benchmark run.

The accuracy of a particular measurement is affected by the combination of all analyzer and components uncertainties, for the ranges and frequency being measured. Consequently, setting and documenting of the correct voltage and ampere ranges is the first and an important configuration step, since these ranges are used for the power analyzer accuracy calculation (which is required for the compensated value calculation).

For example, the formulae below [2-5] illustrates the calculations used to get the power range value of the DBServer subsystem for the TPC-E configuration. The DBServer subsystem is connected to a Yokogawa WT210 Digital Power Meter and the voltage and current range is set to 300 volts, 10 amps respectively.

$$PowerRange = VoltageRange \times CurrentRange \quad (2)$$

$$Vrng = VoltageRange = 300volts \quad (3)$$

$$Irng = CurrentRange = 10Amperes \quad (4)$$

$$\Pr ng = PowerRange = 300volts \times 10Amperes = 3000watts \quad (5)$$

3.5 Power Analyzer Range Settings

The power-range on most power analyzers is indirectly selected by selecting the appropriate voltage and current ranges. The product of the voltage-range and amperes-range is equal to the volt-amps-range, which is also the power-range for real power. Real power is reported by most power analyzers and is the product of the voltage, current, and power-factor. Since the maximum value for power-factor is 1.0 (purely resistive load), this is also equal to the volt-amps.

3.6 Power Analyzer Accuracy Calculations

All measurement devices have some inaccuracies in the readings provided. The TPC has traditionally taken the pessimistic approach regarding accuracy and uncertainties for measurements. Basically this is implemented by accounting for these uncertainties by skewing the reported data to be the most pessimistic value. In the case of measuring the energy utilization for TPC-Energy, the potential errors in the measurements are added to the actual measured values and the compensated values are reported.

The TPC-Energy specification, Clause 6.1.2.6 states that the overall accuracy must be calculated using the vendor provided specifications. This includes the accuracy of the power analyzer, the coupling device (inline, feed-thru, clamp-on), and range selection. For instance, Figure 7 is an excerpt from the Yokogawa WT210 specification chart which provides the accuracy as a *% of reading* and a *% of range*.

The Yokogawa WT210 specification has required values for the uncertainty of the Power Accuracy calculation using *% of reading* (A_{rdg}) and *% of range* (A_{rng}) The calculated Power Accuracy value is used when calculating the Compensated Value and Accuracy Correction Factor (A_{cf}).

Measurement Functions

Parameter		Active power	
System		ng; sum of averages method	
Frequency range		0.5 Hz to 100 kHz	
Crest factor		th minimum effective input)	
Accuracy (three months after calibration)	DC:	DC:	±(0.3% or rdg + 0.2% of rng)*
(Conditions)	0.5 Hz	0.5 Hz ≤ f < 45 Hz:	±(0.3% of rdg + 0.2% of rng)
Temperature: 23±5°C	45	45 Hz ≤ f ≤ 66 Hz:	±(0.1% of rdg + 0.1% of rng)
Humidity: 30-75% RH	66 Hz	66 Hz < f ≤ 1 kHz:	±(0.2% of rdg + 0.2% of rng)
Input waveform: Sinewave	1 kHz < f	1 kHz < f ≤ 10 kHz:	±(0.1% of rdg + 0.3% of rng)
Power factor: cosφ = 1			±((0.067 × (f-1))% of rdg)
In-phase voltage: 0 V DC	10 kHz < f ≤ 1	0 kHz < f ≤ 100 kHz:	±(0.5% of rdg + 0.5% of rng)
Frequency filter: ON at 200 Hz or less			±((0.09 × (f-10))% of rdg)

Fig. 7.

The calculations used for the Accuracy Correction Factor, shown in the formulae below, uses the Power Accuracy and Compensated Value formulae of the TPC-Energy specification, Clause 6.1.2.6 and 6.1.2.7.

$$Ardg = \text{Accuarcy \% of Reading} \tag{6}$$

$$Arng = \text{Accuarcy \% of Range} \tag{7}$$

$$Pavg = \text{Average Power} \tag{8}$$

$$Acf = \text{Accuracy Correction Factor} \tag{9}$$

$$Acf = \frac{Pavg \times Ardg + \Pr ng \times Ardg}{Pavg} + 1 \tag{10}$$

$$Acf = \frac{0.001 \times (1696.34 watts) + 0.001 \times (3000 watts)}{1696.34 watts} + 1 \tag{11}$$

$$Acf = 1.003 \tag{12}$$

The calculated Accuracy Correction Factor is then used to determine the Compensated Value. [13-17] illustrates the Compensated Value Calculation used to compensate for the variations in accuracy of different power analyzers and probe types. The calculated overall accuracy of the device is used to adjust the power measurements by a factor that will ensure that the reported result is conservative.

$$E_m = \text{Measured Energy.} \tag{13}$$

$$A_{cf} = \text{Accuracy Correction Factor .} \tag{14}$$

$$E_{cv} = \text{Compensated Value Energy .} \tag{15}$$

$$E_m = \text{Measured Energy .} \tag{16}$$

$$E_{cv} = A_{cf} \text{ x } E_m . \tag{17}$$

3.7 Power Analyzer Connections to PMU

In addition to the power analyzer settings, during the setup of the TPC-E configuration, it was necessary to group multiple PMU's into a single PMU, depicted in Figure 8.

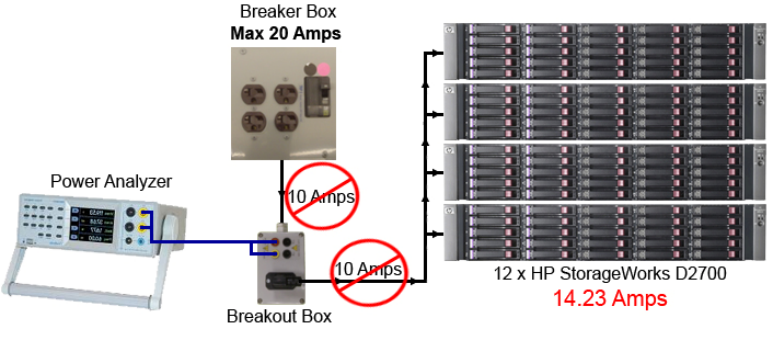

Fig. 8.

Although, the grouping of disk array enclosures may seem simplistic, there are some considerations not to be neglected. Safety is always a primary concern and

ensuring that the maximum current and voltage required by the PMU is within the tolerance of the lowest rated component being used in the energy measuring process.

Initially, the lowest rated component in Figure 8 was incorrectly attributed to the breakout box, which has a 300 volt, 20 amp rating. The actual limiting component was inadvertently overlooked. The power cable was only rated at 10 amps. During the initial full-load measurements the PMU drew 14.23 amps, and caused significant heating of the power cables and the breakout box. In a worst case scenario, this oversight could have resulted in an electrical fire.

4 Energy Measuring System

The TPC-Energy specification, Clause 4.1, defines the Energy Measuring System (EMS), which is the TPC provided software package designed to facilitate the implementation of TPC-Energy measurements. It consists of three software modules which participate in the energy measuring process, depicted in Figure 9 below.

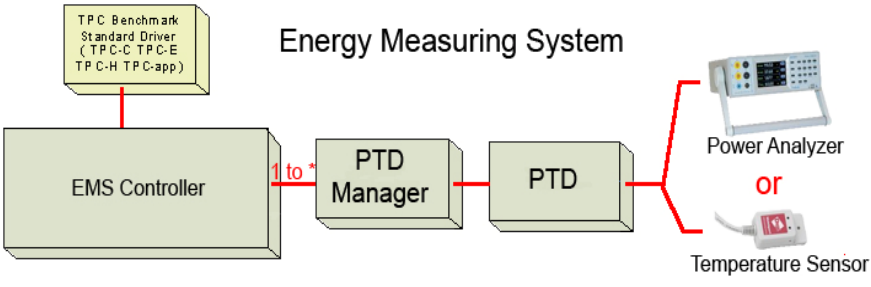

Fig. 9.

During the measuring process a single EMSC instance communicates and controls multiple PTDM's using a socket. Each PTDM is connected to a single instance of the PTD using a socket for communication.

Finally, the PTD's communicate to either a Power Analyzer or Temperature Sensor via an interface adapter (USB to RS232 or USB to GPIB). The EMS modules can be distributed across many controller servers (e.g. remotely or locally), local hosting was preferential.

4.1 Energy Measuring System Controller

The Energy Measuring System Controller (EMSC) is a binary executable, which primary function is to orchestrate the various components involved in TPC-Energy measurements. In addition to connecting to multiple PTDM's, it also provides connections for real-time displays and an interface to the benchmark driver. In other words, it receives configuration and run commands from a TPC benchmark driver (TPC-E in this case) then forwards them to the respective PTDM instances. Similarly, when a PTDM sends power or temperature data, it forwards it to a real-time display

like Microsoft's Performance Monitor. In most cases the default EMSC configuration settings will suffice.

4.2 Power Temperature Daemon Manager

The EMS-PTDM is a binary executable, which primary function is to synchronize, format, log, and forward power or temperature data received from the EMS-PTD to the Energy Measuring System Controller (EMSC). It also sends configuration and run commands to the power analyzer via the PTD.

The TPC-Energy Specification defines secondary metrics in order to allow detailed comparisons of the result, broken down by subsystem. These secondary metrics must be reported in Subsystem-name Watts/Work per unit of time, where Subsystem-name is selected from DBServer, AppServer, Storage and Misc. The PTDM has several command line options in addition to the default settings; in this case the -i [ptdm id] option was used. The -i option allows users to set the Subsystem-name, which is later used for logging and to uniquely identify each PTDM instance. For example, the Database Server Subsystem of SUT would receive the [ptdm id] of "DBServer", and results would be reported in the form of DBServer watts/KtpmC. Instead of using only the Subsystem-name, this setup expounds upon the naming convention by appending the PMU-name. Now, the Database Server Subsystem would be reported in the form Subsystem-PMU name or DBServer-ArrayOne watts/KtmpC. This convention is beneficial when tuning of analysis of specific PMU's is desired. Other than the -i option, the default PTDM configuration settings sufficed.

4.3 Power Temperature Daemon

The Energy Measuring System's Power and Temperature Daemon (EMS-PTD) is a version of the SPEC PTDaemon tool, provided under license from the Standard Performance Evaluation Corporation (SPEC), which includes extensions to meet the requirements of the TPC. It communicates directly to the Power Analyzer or Temperature Probe to obtain the readings. In addition, it sends data to the Energy Measuring System's Power and Temperature Daemon Manager (EMS-PTDM). EMS-PTD configuration settings are available in the EMS package and documentation.

Although, the default "software" configuration settings are many times sufficient, the instrumentation configuration may present several challenges during the energy measurement process. First, the EMS-PTD has the ability to communicate with devices via a Serial or GPIB Interface. Initially, an attempt was made to connect four power analyzers (each using a GPIB controller) to the "controller" server via a four port USB hub. While all devices connected to the EMS-PTD successfully, at least one of the EMS-PTD instances consistently failed to receive reliable power samples during the measuring process. Several additional attempts and configurations did not reliably collect power samples during the measuring process. Once the GPIB Interface (IEEE-488) was used, the EMS-PTD was able to connect to several power analyzers and reliably communicate through a single USB interface port via the National Instruments GBIB to USB controller (GPIB-USB-HS).

4.4 Report Generator

The Energy Measuring System's Report Generator (rgen) is a binary executable that produces standardized reports derived from the configuration and measurement data extracted from the XML logs recorded by the PTDM.

The EMS-Report generator uses options to select the time periods of the specific measurement interval. Using the *-s [start time]* and *-e [end time]* command line options are utilized for this purpose. An example, if the measurement interval started on 2010-06-15 at 22:30:21 and ended on 2010-06-16 at 01:30:21, then the Rgen would be invoked with the command *"rgen -s 2010-06-15T22:30:21 -e 2010-06-16T01:30:21*.

It is highly advantageous to synchronize all the participating systems, including the EMS controller server(s). In this setup the SUT was chosen as the time server and the command, "*net time \\hostname /SET"* was executed on the controller(s) before each benchmark run.

5 Tuning the SUT for Energy Efficiency

Energy consumption of computing systems is directly and indirectly affected by the systems performance. Figure 10 presents a spectrum of various energy reducing techniques [6]. These techniques range from specialized circuits all the way to the application and database level tuning.

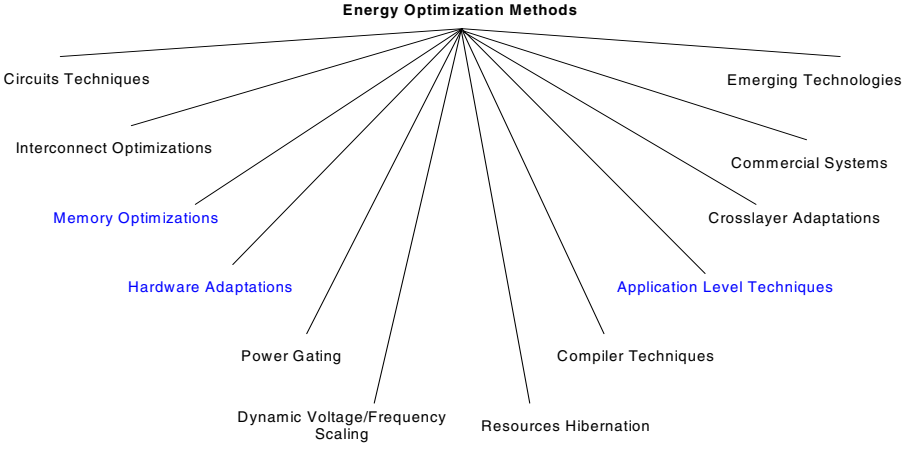

Fig. 10.

Energy consumption management in IT is an evolving field which will continue to be increasingly prominent for many years. While searching for the best-practices and right-sized configurations for optimal energy/performance systems, demand for higher performing servers and increased scale-out implementations continue to drive the technology. Few environments are willing to sacrifice significant amounts of

performance unless there are also significant energy savings. However, it remains to be seen which techniques will be the biggest contributors in solving the energy vs. performance problem.

Among these various methods, memory optimization, hardware adaptation, and application level techniques are implemented at various stages in our quest of tuning the system to provide the most optimal TPC-Energy results.

6 System Configuration

Before configuring a system for performance, we need to know the performance goal. Next step, we plan for the hardware needed to achieve this goal. Among the hardware components, the CPU performance characteristic is the most accepted component to be considered. Information on CPU performance comes from various sources. The first source is the processor vendor's projection of the CPU performance. It provides a reliable and often first hand empirical, and theoretical performance numbers for the CPU. Another source of CPU performance projection comes from in-house modeling efforts. In addition, performance metrics from previous published benchmarks are among the reliable data.

With the initial CPU performance sizing done, we need to look at other crucial components that makes up the benchmarks such as memory, storage to accommodate the I/O rate, network layer between different tiers. Among these, the I/O subsystem is at the top of the list. Both I/O throughput and latency enter the performance sizing picture. Number and type of disks, disk controllers, numbers of I/O slots in the system affect the performance greatly. With the current state of computer system performance, the I/O subsystem is often the bottleneck for some of the TPC benchmarks such as TPC-C and TPC-E.

The network between tiers plays a crucial role in performance goal. It can be a source of bottleneck if it is not planned carefully. In general, TCP/IP protocol provides a reliable, easy-to-adopt network layer. However, others low latency communication layer such as infiniband, iscsi have been used with excellent performance for both network and storage I/O.

The decision between which OS, database, applications to implement is sometime made in parallel with the hardware choice. One thing to remember about system performance is that the role of software is just as important as hardware. Both software and hardware need to be tuned to bring about the optimal performance. Database and OS tuning process are complex and time-consuming. The intricacy of optimizing various parameters all at once is a science and an art. Online resources on how to tune the OS and database are quite plenty and up-to-date.

The next step is configuring the system with the hardware and software list decided above. Here various sizing tools can be used to design the layout with proper number of hardware. For example, HP has the ProLiant Transaction Processing Sizer software to help in design of systems with Microsoft SQL Server database. The tool is intended to properly identify HP server and storage requirements for database transaction processing needs.

When TPC-Energy benchmark is used with other TPC benchmarks, proper current, amperage sizing for various DB server, application servers, network switches, and storage enclosures needed serious consideration and isolation from the surrounding for the ease of benchmarking. Proper and correct usage of the power cables and connectors is a must to avoid accident. In system tuning, there are several stages involved:

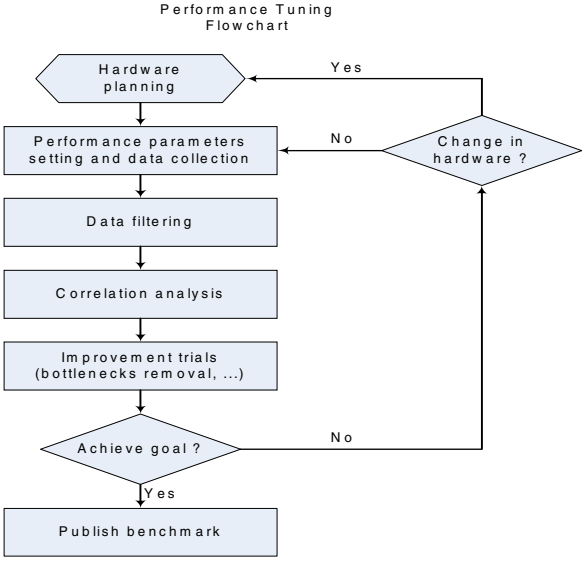

Fig. 11.

After stage 5, if the performance goal is met, the benchmark is slated for eventual publication if it meets the compliance criteria, such as ACID tests, etc. required by the TPC specifications. If the performance, price/performance, or energy/performance goals are not met, an evaluation of the configuration of the hardware and software analyzed and changed based on the findings.

Tuning all three parameters of performance, energy, and price (SUT price) of the system is a complex and challenging journey. Considering the two examples shown in Error! Reference source not found., the performance is optimized while the energy consumption and price can be reduced. The goal is to reduce the energy and price, thus make the PEP volume (performance, energy, price volume of the right triangular pyramid) as small as possible.

However, Figure 12 is only a special case where performance is already optimized. The overall goal of tuning in performance, energy and price 3-D landscape is to reduced the area of base, i.e. the product of Energy Consumption and Price while increase the Performance height of PEP volume.

A more intuitive picture is to couple the Energy and Price dimensions with Performance as shown in Figure 13.

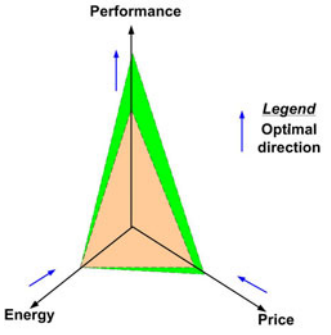

Fig. 12.

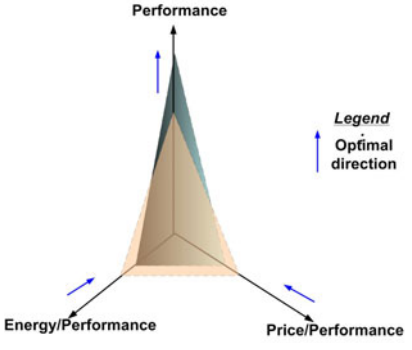

Fig. 13.

Scaling Energy and Price dimensions to Performance gives a more intuitive picture for optimization. A formula [18-20] for the scaled PEP volume is

$$\textit{Scaled PEP volume} = 1/3 \text{ (Area of Base) x (Height)} . \tag{18}$$

$$= 1/3 \text{ (Energy/Performance x Price/Performance) x (Performance)} . \tag{19}$$

$$= 1/3 \text{ (Energy x Price) / Performance} . \tag{20}$$

Thus, the goal for tuning is clearly to reduce the scaled PEP volume of Figure 13. Here, an increase in performance also helps reducing the other two dimensions thus help reducing/optimizing the scaled PEP volume at the same time.

7 Experimental Data

Data was collected on four different configurations. The SUT is an HP-DL580G7-4P with SAS or SSD drives. The configuration with SAS drives is the same as the one listed in Figure 2. For example, SUT-SAS-Power Save configuration means the SUT has SAS drives running in Power Save mode. Similarly, SUT-SSD-Power Save indicates that the SUT is running with SSD in Power Save mode. In the SSD configuration

the SUT has 320 drives in RAID0+1 using a total of 16 + 1 storage cabinets, where the additional cabinet is for database backup purposes. The pricing is similar to that of TPC-Pricing rules and the TPC-E workload was used. Some of the energy data for subsystems such as clients, monitors, miscellaneous are taken and/or extrapolated from collected data.

Figure 14 utilizes equation-20 to determine the optimal Scaled PEP volume. Among the four configuration shown, the least optimal is the SUT with SAS drives running in Power Save mode. Here, the (Energy/tps, Price/tps, Performance) are measured at (6.37 watts/tps, 384.93 USD/tps, 1800 tps), see Figure 15.

The optimal configuration is the SUT with SSD drives running in High Performance mode. Here, the (Energy/tps, Price/tps, Performance) are measured at (1.72 watts/tps, 440.51 USD/tps, 2102 tps).

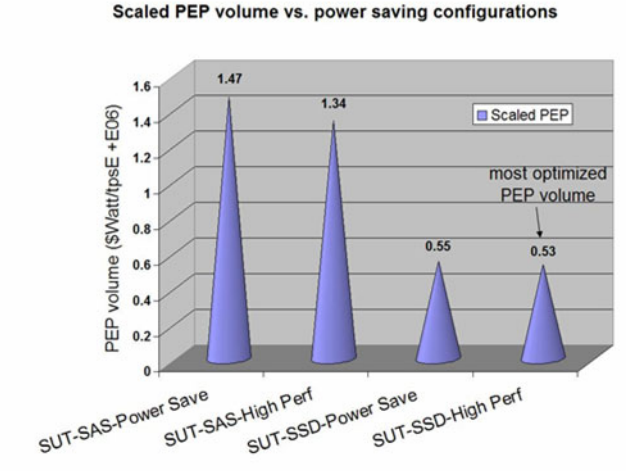

Fig. 14.

8 Conclusion

The Transaction Performance Processing Council has provided benchmarks which enable the agnostic measurement of database systems performance and price/performance, and now energy/performance. All major hardware and software vendors participate and/or utilize TPC Benchmarks as part of the evaluation of their products. The corporate IT customers are demanding higher performance and less energy consumption at reduced costs. The TPC-Energy results provide the most comprehensive evaluation of this growing concern with a level playing field for IT vendors to compete. Additional aspects of the TPC benchmarks are also significant when using the published results.

Acknowledgements

The authors acknowledge the significant contributions of HP employees, Daniel Pol and David Adams, for their long hours of work and blazing this new trail to publish the first TPC-Energy Benchmarks.

The TPC Benchmark specification and software were developed by the members of TPC-Energy committee. Their contributions are greatly appreciated and the authors are thankful to them for their dedication and work to provide this benchmark.

The EMS-PTD is based on the SPEC PTDaemon tool, and is provided to the TPC under license from the Standard Performance Evaluation Corporation (SPEC).

The TPC auditors, Lorna Livingtree and Steve Barrish, also contributed to the publication of these first TPC-Energy benchmarks. Thank you for all the hard questions, turning over all the proverbial rocks, and the late nights in the lab, ensuring that the results are TPC compliant.

References

1. Transaction Performance Council website (TPC), http://www.tpc.org
2. Standard Performance Evaluation Corporation website (SPEC), http://www.spec.org
3. TPC Invitation to Join, http://www.tpc.org/information/about/join.asp
4. TPC-E and TPC-Energy Specifications, http://www.tpc.org
5. Yokogawa Electric Corporation, http://tmi.yokogawa.com/
6. Venkatachalam, V., Franz, M.: Power Reduction Techniques For Microprocessor Systems. ACM Computing Surveys 37(3), 195–237 (2005)

Using Solid State Drives as a Mid-Tier Cache in Enterprise Database OLTP Applications

Badriddine M. Khessib, Kushagra Vaid, Sriram Sankar, and Chengliang Zhang

Microsoft Corporation, Redmond, WA, USA
{bkhessib,kvaid,srsankar,chengzh}@Microsoft.com

Abstract. When originally introduced, flash based solid state drives (SSD) exhibited a very high random read throughput with low sub-millisecond latencies. However, in addition to their steep prices, SSDs suffered from slow write rates and reliability concerns related to cell wear. For these reasons, they were relegated to a niche status in the consumer and personal computer market. Since then, several architectural enhancements have been introduced that led to a substantial increase in random write operations as well as a reasonable improvement in reliability. From a purely performance point of view, these high I/O rates and improved reliability make the SSDs an ideal choice for enterprise On-Line Transaction Processing (OLTP) applications. However, from a price/performance point of view, the case for SSDs may not be clear. Enterprise class SSD Price/GB, continues to be at least 10x higher than conventional magnetic hard disk drives (HDD) despite considerable drop in Flash chip prices.

We show that a complete replacement of traditional HDDs with SSDs is not cost effective. Further, we demonstrate that the most cost efficient use of SSDs for OLTP workloads is as an intermediate persistent cache that sits between conventional HDDs and memory, thus forming a three-level memory hierarchy. We also discuss two implementations of such cache: hardware or software. For the software approach, we discuss our implementation of such a cache in an in-house database system. We also describe off-the shelf hardware solutions. We will develop a Total Cost of Ownership (TCO) model for All-SSD and All-HDD configurations. We will also come up with a modified OLTP benchmark that can scale IO density to validate this model. We will also show how such SSD cache implementations could increase the performance of OLTP applications while reducing the overall system cost.

Keywords: Solid State Disks, OLTP performance, SSD caching, DBMS buffer pool management.

1 Introduction

The recent performance and reliability improvements of Solid State Drives (SSD) have opened a window of opportunity for this storage technology in the enterprise database market [1] [2]. SSDs provide read/write IO rates that are 100 times faster than a conventional Hard Disk Drives (HDD) while maintaining a comparable or

R. Nambiar and M. Poess (Eds.): TPCTC 2010, LNCS 6417, pp. 153–168, 2011.

slightly higher sequential read/write bandwidth. However, prices/GB of SSDs is still at least 10 times higher than that of conventional HDDs. This huge price gap introduces several challenges in directly replacing a HDD-based storage infrastructure with an SSD based storage solution.

In the enterprise market, for SSD based storage solutions to be common place, it has to meet or beat the price/performance of conventional HDD based solutions. While some studies predict the prices to continue to slide at a faster pace than HDD prices, others doubt it will happen any time soon. For example, IDC, in a report released in 2007 [3], predicts the price of SSDs to continue to decline at an annual rate of 50% till year 2012. Others [4], argue that when supply-side cost factors of the two technologies are taken into account, SSD based solutions will not achieve $/GB parity with HDD based solution any time soon.

We believe that the introduction of SSDs in the enterprise storage infrastructure represents a qualitative shift that has the potential to affect how IO intensive applications are designed. In particular, DBMSs have some architectural decisions to make. The first approach is to treat the storage layer as a homogenous and flat extent of disk blocks whether it is SSD or HDDs based. This approach may garner some performance gains at high cost. The second approach is to take advantage of the high random read/write rates at low sub-millisecond latency of SSD based solutions while trying to fray their associated high cost. It is our belief that in this approach the DBMS has to treat the SSDs as an intermediate layer in memory hierarchy bridging the gap between the high speed memory and the very slow HDDs.

1.1 SSDs as a Mid-Tier General Purpose Cache in DBMSs

A quick look at the current storage landscape shows the two extremes that dominate the market. On one hand, DRAM is extremely fast (~100 Nano-second access time), but relatively expensive (~36$/GB). At the other side of this dichotomy is SATA disks (Table 1). SATA, which accounts for more than 90% of the disk storage sold annually [4], is relatively cheap ($0.15/GB) but very slow. Between this duopoly, there are several storage solutions that strive for existence and attention. The challenge with all these storage technologies is they are neither faster than DRAM nor cheaper than SATA.

Today, the SAS 7.2K and SATA 7.2K solutions are converging in terms of price and performance. This is expected to increase with the SATA 3.0 adoption. We believe that eventually, SATA 7.2K may end up replacing SAS 7.2K in the not distant future. In the rest of the paper whenever we refer to SAS, we mean the 10K and 15K flavors.

Currently, SAS storage solution account for ~5% of the total storage sold [3]. In particular, SAS disks are heavily used in enterprise database applications where random I/O throughput is essential to the overall system performance. A quick look at the TPC-C and TPC-E benchmark publications [5] proves this point. Compared to SSDs, the $/GB of SAS disks is anywhere from 10X-15X cheaper while the $/Perf is

Table 1. Cost vs. Performance of different storage topologies. DRAM performance is measured by the number of 8 byte reads/sec. SSD and HDD performance is measured by the number of 8K random reads.

Storage Type	Size(GB)	Price ($)	Perf	$/GB	$/Perf	Watts
DRAM	4	143	1000000	35.75	0.000143	6
SSD	120	1244	10000	10.37	0.12	2
SAS(15K)	300	216	200	0.72	1.08	14
SAS(10K)	300	186	150	0.62	1.24	8
SAS(7.2K)	2000	293	100	0.15	2.93	5
SATA(7.2K)	2000	293	100	0.15	2.93	5

10X-15X more expensive. In this paper we will show that by introducing SSD as a mid-tier cache, we can increase the performance of SAS based solutions.

In this paper we will discuss two ways to implement three-tiered memory architecture in a DBMS: a hardware solution and a software-based one. In addition, we will use this prototype to run a customized OLTP benchmark with different IO densities to determine how these approaches (hardware and software-based SSD caching) compares to the conventional HDD and complete SSD solutions.

1.2 Contribution

The recent improvement in SSD read/write throughput and improved reliability make this technology a compelling enterprise storage solution. However, their $/GB make it unattractive as a complete replacement for HDD based solutions. We propose to use SSDs as a persistent cache in a 3-level memory hierarchy: RAM-SSD-HDD.

Our contributions in this paper are:

- We develop a TCO model for comparing the cost effectiveness of an All-SSD approach compared to an All-HDD solution.
- We implement SSD-based persistent cache in an in-house relational DMBS. We use such prototype to collect performance numbers of different OLTP configurations with different IO densities. We will also pick up a commercially available hardware caching solution and collect performance numbers for the same OLTP configurations.
- We demonstrate that the introduction of SSDs as a mid-tier cache will increase performance and reduce cost in most enterprise OLTP workloads. This approach, while taking advantage of the SSD's high IO throughput also minimizes the cost associated with it.
- We discuss the future implications of such persistent cache on different aspects of the DBMS such as checkpointing, recovery, and backup/restore.

The rest of the paper is organized as follows. In section 2 we will cover related work in this area and the introduction of SSDs in enterprise in particular. Section 3 will be

devoted to discussing the software and hardware implementation of the SSD cache. In section 4, we will describe the TCO model of SSD vs. HDD and discuss the different parameters involved in this model. Based on that analysis, we will design an OLTP benchmark that will allow us to control those parameters. We will conclude this section by describing the system we used to collect the performance numbers. In section 5 we will discuss the results using HDD only, SSD only, Software SSD caching, and Hardware SSD caching. In section 6, we will discuss issues and opportunities associated with persistent cache. Section 7 will include our concluding remarks.

2 Background and Related Work

Solid State Drives:
Solid State Drives are becoming predominantly important in the storage research community. Decreasing prices and increasing volume in different markets have enabled more technologies in this domain in both research and industry segments. Companies manufacture both PCI-e based devices and also SATA devices and this has led to several interesting architectures. SSDs can play a role as hot page storage as proposed in [6] for databases, or can replace the main memory buffer cache as proposed in [7]. SSDs can also be used to store file system metadata [8], hash based indices [9] and also an entire DBMS in Flash [13].

Solid State Drives in Enterprise:
There has been active research for using SSDs in enterprises. Solid state disks for enterprise databases was proposed in [1] where magnetic disks could be replaced with flash based memory for transaction log, rollback segments and temporary table spaces. There are industry whitepapers [10] which discuss the performance and reliability requirements for enterprise SSDs. The most closely related paper to our research is [11], where the authors analyze the tradeoffs of using SSDs in enterprises and evaluate SSD as a caching device as well. They show with an analysis of enterprise traces that only 10% of the workloads would benefit from having SSDs. Though we agree with the author's conclusions that SSDs would be useful only if there are data that can be cached in that layer, we believe that a trace based analysis does not change the IO distribution based on a new storage hierarchy. Datacenter applications that are already optimized to work on data in memory have a different distribution than one that would work on SSD cache. Hence, in our work we tune benchmarks based on our application profile and vary IO densities and then run an end-to-end system evaluation.

3 Implementation of an SSD Cache

We used two approaches to implement an SSD cache. The first is a software solution where we took an in house DBMS and modify it to take advantage of the SSD as cache. In the second approach we use off-the-shelf hardware solution that will allow

an application to transparently take advantage of the SSD cache without major re-writes. Below, we will discuss both approaches and discuss their respective merits and drawbacks.

3.1 Software SSD Solution

We modified an in-house DBMS to take advantage of SSD as cache. As any modern DBMS, it has a memory manager that allocates the physical memory available in the system and breaks it into 8KB buffers. These buffers are initially added to the free buffer pool. Whenever the DBMS tries to read a page from disk the memory manager will allocate a free buffer and read the disk page into it. In case, there is no free buffer to read the disk page into, we free an allocated buffer using an LRU scheme. Hence, the pool of allocated buffer is essentially a cache for the HDD pages. We use a hashing mechanism to quickly locate a page within this cache (see Figure 1).

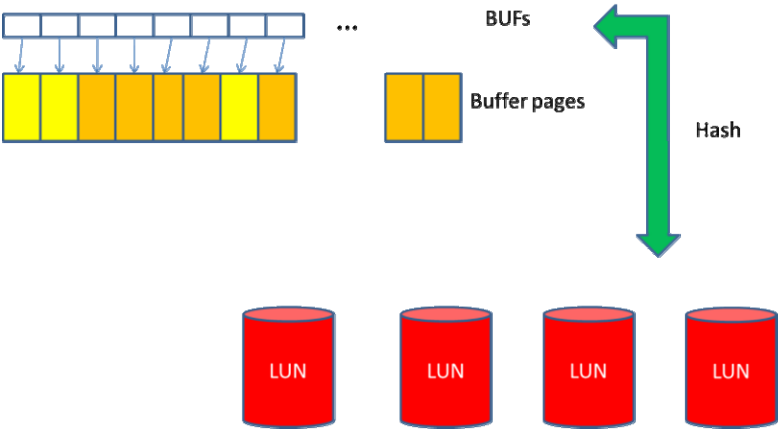

Fig. 1. Typical Working of DBMS memory manager

We added a look-aside SSD cache to the above memory manager (see Figure 2). Each time a disk page is requested, we look it up in the memory cache. If the page is found, the memory manager returns it. However, the disk page is not found, we look up the look-aside SSD cache. If the disk page is found, the memory manager will load it into the memory cache. In case the disk page is not in the SSD cache, the memory manager will load the page into the memory cache. Whenever a disk page is evicted from the memory cache due to memory pressure and the disk page is already is already in the SSD cache, nothing further is done. However, if the disk page is not in the SSD cache and there is a free SSD page, the disk page will be written to that free SSD page. In the event there is no free SSD page, we select an eviction candidate from SSD cache using an LRU based scheme (There can be multiple schemes here similar to [14]). We then compare the timestamps on the both eviction candidates. If the timestamp on the memory eviction candidate is more recent, we then write it to the SSD page.

Because the hash tables and all the ancillary data structures that help manage the SSD cache is in RAM, the SSD cache is not persistent. If the persistence property is desired, then having an emergency backup power source for the server will allow storing these data structures in SSD in case of an unforeseen power loss. When power is restored and the server is brought back online, these data structures will be read back from the SSD and the cache will be accessible again.

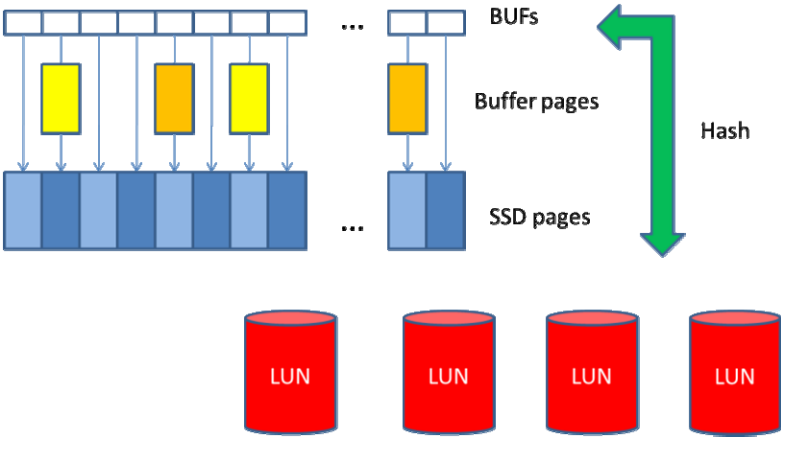

Fig. 2. DBMS memory manager with SSD pages

3.2 Hardware Solution

Another approach to SSD caching is to use RAID controllers with SSD caching feature. PMC-Sierra's Adaptec hardware accelerator [12] is a publicly available RAID controller that can work with SSD caching. These controllers leverage their presence in the data path to provide several cache levels to speed up IO operations. They also provide memory cache in addition to an SSD cache. They identify frequently accessed data and copy it to attached SSD cache pool for faster access. Unlike, the software solution which requires modifications to applications to enable SSD caching, the hardware solution provide a transparent and seamless solution. Unlike the software solution, these RAID controllers are equipped with battery backup. Hence, cache persistence is supported out of the box. Some of these RAID controllers provide other useful features such hot plug and failure isolation. Hot plug allows the removal and addition of SSD disks to the SSD cache pool without requiring a system shutdown. Failure isolation, on the other hand, enables the SSD cache and the system in general to continue operating even when some of the SSDs in the SSD cache pool fail. When the controller detects a disk failure, it will automatically take the disk out of the SSD pool. Similarly, a system administrator could add/remove disks to the SSD cache pool without causing the SSD cache to go offline.

4 Experiment Methodology and Infrastructure

We will start this section by discussing the cost model of an all-SSD solution and the all-HDD solution. The cost model discussion will point out a dominant factor that will decide which solution will more cost effective: IO density. Based on this finding, we will transition into the discussion of a benchmark that will allow us to simulate workloads with different IO densities. Finally we will discuss with detail the system configuration of a typical production server that was used to collect all the performance numbers.

4.1 The Cost Model

From Table 1, $/Perf of a 10K RPM SAS based HDD is higher than the $/GB of the same drive. Therefore, we believe that the cost structure of an HDD based solution will be dominated by the former. Since power is becoming the main constraint in a data center environment [15], we added the cost of power to the cost equation as well.

$$Cost_{HDD} = IOPS * \$/IOPS_{HDD} + Power_{HDD} * \$/Watt \quad (1)$$

In the SSD case, however, the $/GB dominates the $/Perf. Hence, the cost structure of an SSD solution will be based on $/GB.

$$Cost_{SSD} = GB * \$/GB_{SSD} + Power_{SSD} * \$/Watt \quad (2)$$

For the SSD solution to become economically viable,

$$CostHDD > CostSSD \quad (3)$$

$$IOPS * \$/IOPS_{HDD} + Power_{HDD} * \$/Watt > GB * \$/GB_{SSD} + Power_{SSD} * \$/Watt \quad (4)$$

After few transformations, we end up with the following inequality.

$$(IOPS/GB) * \$/IOPS_{HDD} + (Power_{HDD}/GB - Power_{SSD}/GB) * \$/Watt > \$/GB_{SSD} \quad (5)$$

We make few notes and definitions before we proceed. IOPS/GB is called ***IO Density***, which we will refer to as IOD in the rest of the paper. IOD is independent of the storage technology and is a property of the workload [17]. $Power_{HDD}/GB$ (PD_{HDD}) and $Power_{SSD}/GB$ (PD_{SSD}) are power densities of HDD and SSD respectively. $PD_{HDD} - PD_{SSD}$ is the *power density delta* between HDD and SSD, which we will refer to as PD_{Δ} in the rest of the power. Using all the above, the new cost model will be:

$$IOD * \$/IOPS_{HDD} + PD_{\Delta} * \$/Watt > \$/GB_{SSD} \quad (6)$$

Our production system uses 10K rpm SAS drives. Using Table 1, the values for $\$/IOPS_{HDD}$, PD_{Δ}, and $\$/GB_{SSD}$ are $1.24, 0.01 Watt/GB, and $10.37, respectively. The $/Watt is the total cost of power which include the cost of actual power consumed plus the data center infrastructure cost. The value of this parameter is pegged at $10/Watt. Substituting all the values in the equation above and solving for IOD, we get: **IOD > 8.28**. This says that a complete SSD solution will not be cost effective unless the IO density of the workload is 8.28 or higher. In this paper we will attempt to answer two questions. First, does SSD caching (hardware or software) helps to achieve performance gains for IO densities less than 8.28? Second, how does SSD caching perform compared to the total SSD solution for IO densities higher than 8.28? (Given assumptions about the RAID setup that we present in Section 4.3)

To answer these questions, we have to come up with a benchmark that is scalable, can easily generate different IO densities, and mimic the behavior to our OLTP application in the cloud.

4.2 Benchmark Description

With the above goals in mind, we started with the standard TPC-E benchmark [5]. We created a 50K customer database to fill the data partition of the system (see next section for details). Then, we tried to generate different IO densities by only exercising a subset of the users (10K, 20K, 30K, 40K, and 50K). We immediately hit few issues with this configuration. First, the workload did not scale very well. With 10K and 20K customers we notice sizeable lock contention resulting in poor performance. Second, the initial IO density is still in double digit.

To tackle the scalability issue, we had to increase the number of users on the same data partition. In order to do that, we had to reduce the number of disk space allotted to every TPC-E customer. Our TPC-E benchmark kit allows that by using two switches: scale factor and trading days. A legal (auditable) TPC-E database requires values of 500 and 300 respectively for both knobs. With those values, a 1000 customer database takes around 10 GB of disk space. In our modified TPC-E benchmark, we used 2000 and 30 for scale factor and trading days. With these values, a 1000 customer databases consumes around 400 MB of disk space. Hence, we have a reduction of disk space/customer by a factor of 25. This big reduction of disk space requirement per customer, allowed us to build a 1 million customer TPC-E database.

Second, we modified the TPC-E transaction mix to allow I/O to scale by the number of customers. The changes to transaction mix are as follows:

1. Market watch has a weight of 0
2. Trade lookup frame 4 has 100% weight
3. Trade update “Read rows” set to 2
4. Trade update “update rows” set to 1
5. No data maintenance transactions

These changes make the new workload closer to the actual production workloads in two ways. First, in real online OLTP applications, the server hosts millions of users.

However, only a subset of users is active at any time. Second, IO densities of online OLTP applications tend to be very low [11] [17]. It also enables us to scale the IO density by varying the active customer count. Table 2 shows the IO densities various customer counts that will be used in all our experiments.

Table 2. Customer count and corresponding IO densities

Customer count	250K	500K	750K	1000K
IO density	2.27	7.31	16.30	25.74

4.3 System Configuration

Table 3 shows the components of a typical production server which we used for performance measurements.

Table 3. System Configuration

System configuration	
Platform	HP ProLiant DL380 2U Model
CPU	2 x Intel Xeon L5520 2.26GHz, 8MB L3 Cache, 60W
Memory	12 x 4GB PC3-10600R (48GB total)
HDDs	20 x 146GB 10K SAS 2.5" , 5 x 300GB 10K SAS 2.5"
Controller	PMC-Sierra's Adaptec hardware accelerator
Power Supply	2 x 750W Gold N+1 Redundant Power
Network	Single Dual Port Embedded Intel NIC

Out of the 24 SAS disks, 14 146GB disks are short stroked in a RAID10 configuration to form a 600GB (H drive) data partition (see Figure 3). We also used 4 146GB disks in RAID10 configuration to make the temporary database partition (T drive). The 5 300GB SAS disks are grouped together in a RAID5 configuration to make the 150GB log and 760GB backup partition (O and E, respectively).

For the All-SSD configuration, we replaced the 14 146GB data disks with 12 HP 120GB SSD disks. We grouped these SSDs in a RAID10 configuration to form a 600GB data partition. For the SSD cache we used 4 Intel X25 SLC 30GB disks.

5 Experiment Methodology and Infrastructure

In this section, we will discuss the experimental results of the different configurations: All HDD (*Baseline*), All SSD, hardware caching, and software caching. We will also compare the TCO score of each solution.

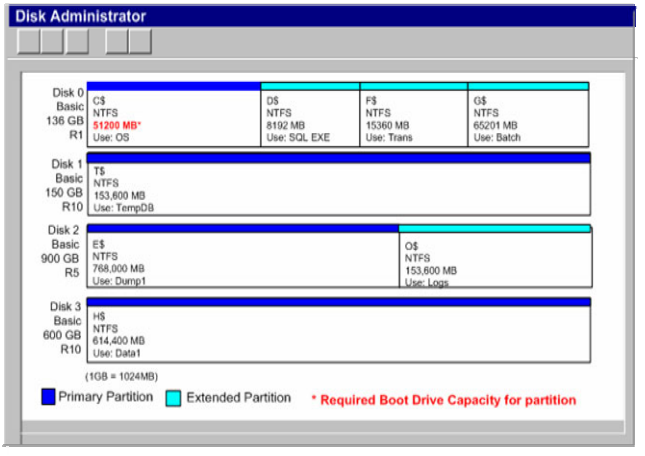

Fig. 3. Disk layout for System under Test (SUT)

5.1 All SSD vs. All HDD Solution

We ran our benchmark with its four flavors (representing different IO densities) on an All-HDD (*Baseline*) and an All-SSD configuration. Figure 4 shows the performance (tpsE) of both configurations. The 250K-user (IOD 2.27) showed no gain when substituting HDDs with SSDs, since the memory housed most of the dataset in both configurations. The remaining 3 configurations showed substantive gains ranging from 25% to 536%, when more requests had to be served from the disk configuration. Note however that the system price of the SSD solution is significantly higher than

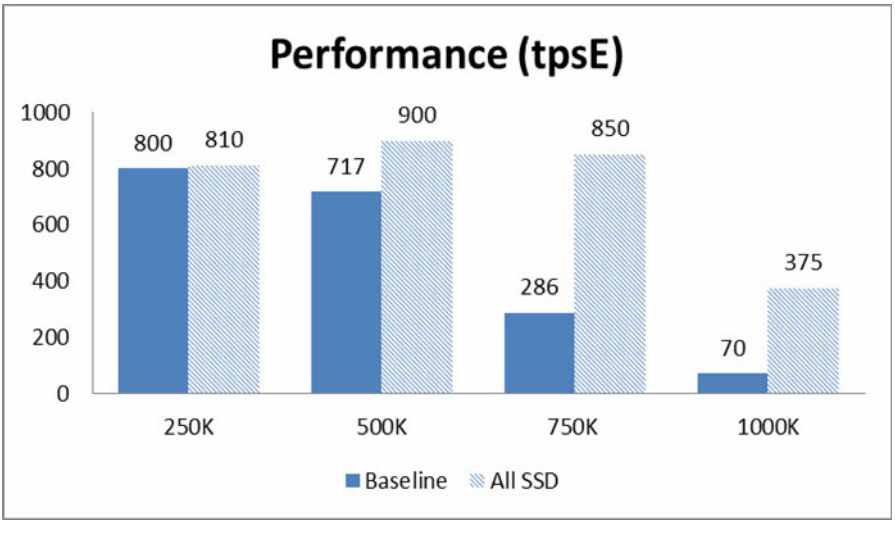

Fig. 4. Performance (tpsE) of Baseline compared to an All SSD solution (All SSD better in all cases)

the All HDD - Baseline solution. Figure 5 shows the corresponding TCO numbers. We calculate the tpsE/$/W which essentially gives the performance:cost benefit. Runs with IO densities less than 8.28 show that the HDD configuration is cheaper. The SSD configuration comes ahead with runs with IO density of 8.28 or higher. This is an experimental validation of our TCO model discussed in section 4.1.

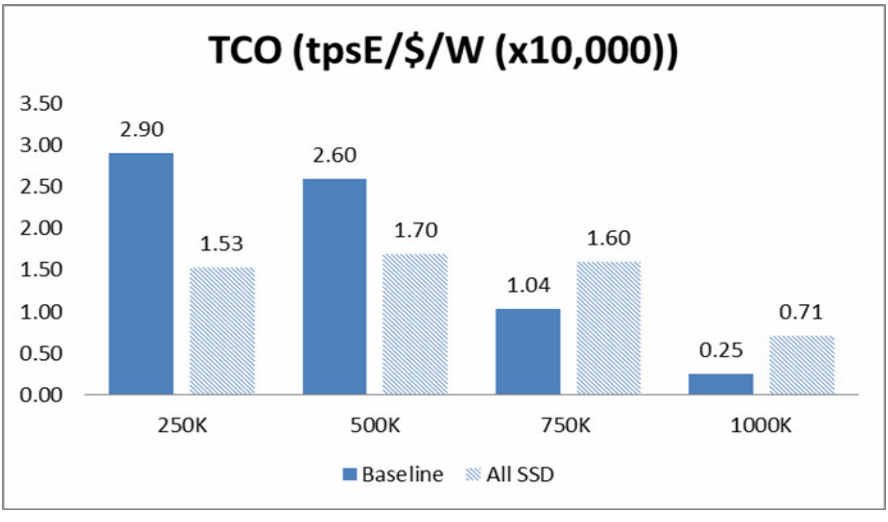

Fig. 5. TCO comparison between Baseline and All SSD solution (Higher value is better)

5.2 Hardware SSD Caching Solution

As discussed in section 4.3, we used PMC-Sierra's Adaptec hardware accelerator solution [12]. For each benchmark flavor, we run with four different cache sizes: 32GB, 64GB, 96GB, and 128GB. This will allow us to find the optimal cache size (and best TCO) for every run. Figure 6 shows performance results of the Hardware SSD caching run under different SSD sizes. As in the All-SSD configuration, 250K run did not show any gains from hardware caching. The 500K run shows gains of as much as 25%. It also shows that cache sizes bigger than 96GB will not improve performance. The remaining two runs show substantial performance gains of 286% and 514%, respectively. These results are very impressive as they are almost identical to the All-SSD solution.

Since the hardware caching solution is cheaper than an All SSD solution, the TCO numbers are much better for the SSD caching solution (see Figure 7). Another interesting finding is that the SSD caching solution helps for IO densities lower than 8.28 also. Figure 8 shows that the hardware SSD caching solution (with 64GB cache size) has better TCO than an All-HDD solution for the 500K run. In comparison, the All-SSD solution failed at achieving this result for the 500K run. For the 750K and 1000K runs, the hardware SSD caching solution beats both All-HDD and All-SSD solutions in terms of normalized TCO.

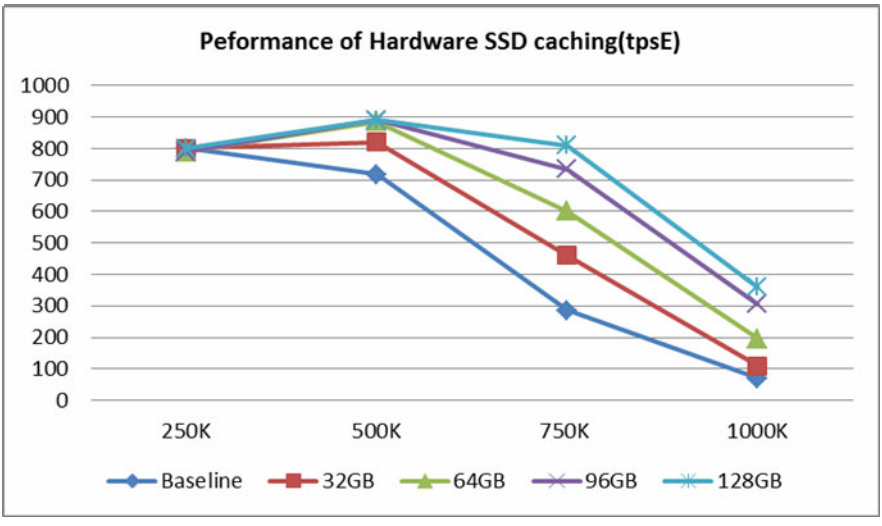

Fig. 6. Hardware SSD caching solution shows gains in tpsE over Baseline

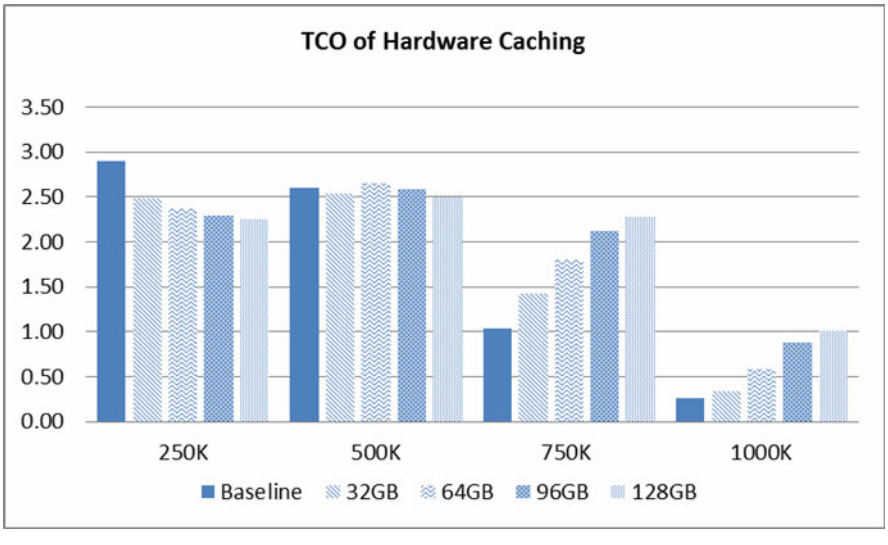

Fig. 7. Hardware SSD caching shows better gains in TCO (Higher value is better)

5.3 Software SSD Caching Solution

We used the modified in-house DBMS as described in section 3.1 to collect performance numbers. As in the Hardware SSD caching solutions, for each flavor of the benchmark, we used four sizes for the SSD cache, 32GB, 64GB, 96GB, and 128GB. Since the SSD caching is tightly integrated with the memory manager, we expect the Software SSD caching to beat the hardware caching solution. This experiment will

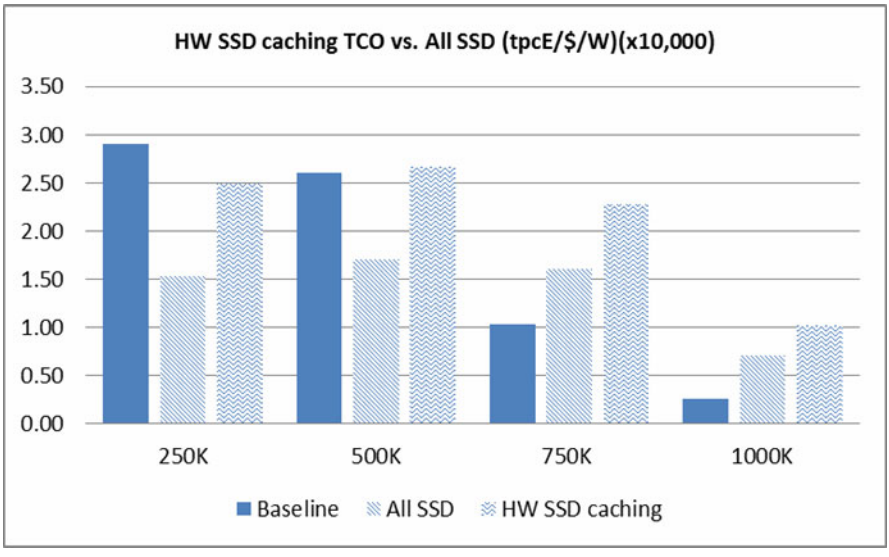

Fig. 8. Hardware SSD caching is better than both baseline and All SSD solution for 500K, 750K and 1000K cases

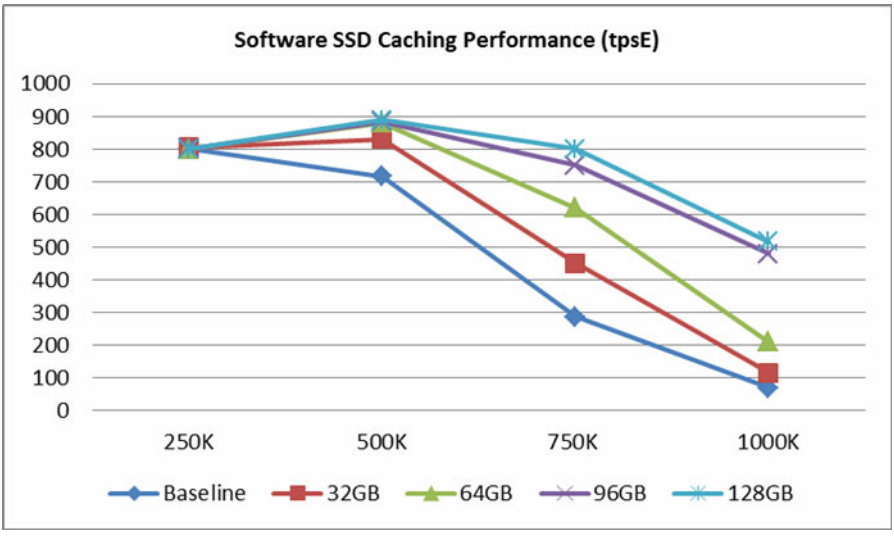

Fig. 9. Software SSD caching shows better performance than Baseline

show us the extent of these gains, if any, and the circumstances under which these gains are realizable. Figure 9 shows the performance numbers of the software SSD caching solution. The performance of the SSD software caching solution is very similar to that of the hardware SSD solution in three out of the four runs: 250K, 500K, and 750K. In the 1000K run, however, the software SSD caching solution is 46% better

than the hardware SSD aching solution. The software SSD solution has a better TCO numbers than hardware solution due to a better performance (as is the case in the 1000K run) and lower price due to the lack of a Hardware Accelerator (see Figure 10 & Figure 11). We see that in all cases the Software SSD Caching solution performs better in terms of performance and TCO than the all Hardware Caching solution. However, it includes the cost and complexity if the software. Hence, it has a lower TCO than the baseline in the 250K case, where caching is not used much. However, for all the other cases (500K, 750K and 1000K), the software caching performs better than the baseline as well.

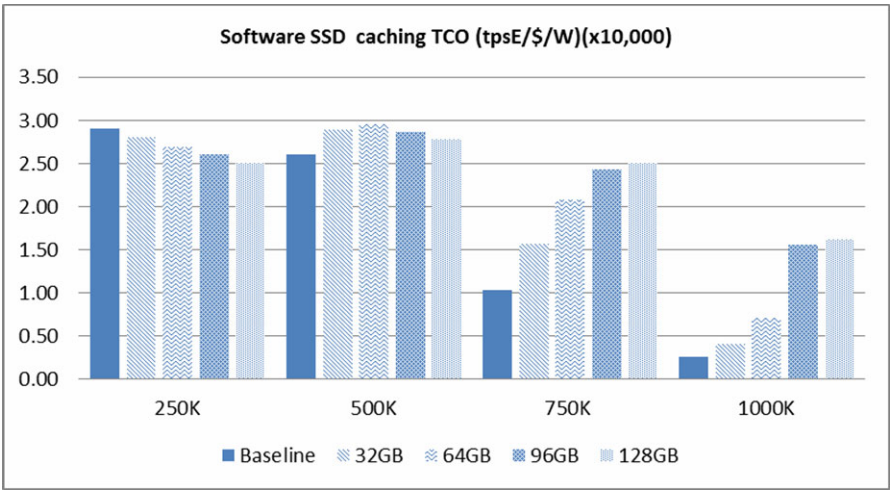

Fig. 10. Software SSD caching has better TCO than baseline in 500K, 750K and 1000K cases

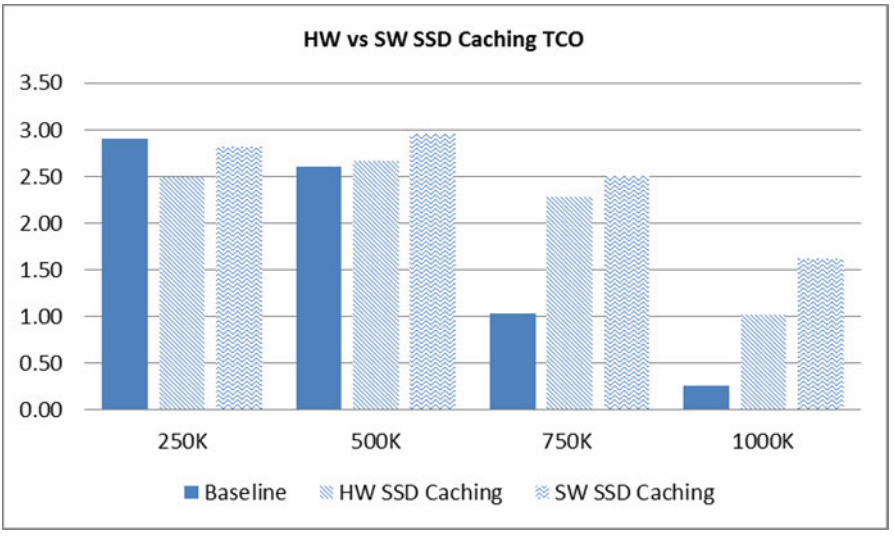

Fig. 11. Software SSD caching is better than HW SSD caching in all cases

6 Future Work

In both software and hardware SSD caching implementation discussed in this paper, a write through caching approach was implemented. There are few reasons for this choice. First, in the software SSD caching, the data structure that maps database disk pages to SSD pages resides in memory. Since memory is volatile, these data structures will be lost leading to data loss. Second, there is a worry about the SSD endurance caused by cell wear. A write-through reduces writes to the SSD cache, ensuring a longer life span. Third, there is a perception issue. SSD is a new technology and there is still some mistrust or "wait and see" attitude. There is a lot of opportunity to address these challenges with new architectural enhancements for SSD solutions.

Impact on DBMS: The PMC-Sierra's Adaptec hardware accelerator has a battery backup. That in conjunction with better endurance numbers due to wear leveling techniques should make a write-back cache a reality. Such persistent cache could have huge implication on checkpointing and recovery in a relational DBMS. For example, in a system with 48GB, assuming 50% of the pages are dirty and a checkpoint is issued every 30 minutes, checkpointing will generate as much as 2000 writes/second. Given, that most enterprise systems use RAID10 for data volumes, this write rate will translate into 4000 IOPS. Given that a 10K rpm SAS HDD supports about 150 IOPS while maintaining a decent latency, the system will need a minimum of 26 disks to support the checkpoint write rates. Having a persistent write-back SSD cache that can absorb checkpoint writes will significantly reduce the need for HDD spindles.

On the workload front, our work was primarily focused on OLTP workloads. Other areas that merit further investigation are DSS workloads and other non-database workloads such as file servers and web servers. We believe that an extensive workload characterization, benchmark tuning and TCO based selection similar to the methodology in this paper would quantify the benefits of SSD based solution in such environments.

7 Conclusion

Despite the improvement in both performance and reliability combined with the steady drop in price, SSDs are still not an outright cost effective alternative to HDDs. We presented a TCO model that states that workloads with IO densities of 8.28 or higher will gain from an all-SDD replacement of HDDs. This model was validated by our experimental results. We tuned the TPC-E benchmark to exercise different IO densities that reflect a spectrum of production workloads. Further, we showed that both hardware and software SSD caching are able to meet the performance numbers of an All-SSD solution while handily winning in the TCO analysis. Another advantage of SSD caching, both hardware and software, is their viability in terms of pure performance as well as TCO for IO densities lower than 8.28 where an All-SSD solution is extremely cost prohibitive. We also show that a software based SSD caching solution would yield the best TCO, thereby leading to datacenter energy and cost savings.

References

1. Lee, S.-W., Moon, B., Park, C., Kim, J.-M., Kim, S.-W.: A Case for Flash Memory SSD in Enterprise Database Applications. In: Proceedings of the ACM SIGMOD, pp. 1075–1086 (2008)
2. Tsirogiannis, D., Harizopoulos, S., Shah, M.A., Wiener, J.L., Graefe, G.: Query processing techniques for solid state drives. In: Proceedings of the 35th SIGMOD International Conference on Management of Data, Providence, Rhode Island, USA, June 29-July 02 (2009)
3. Janukowicz, J., Reinsel, D., Rydning, J.: Worldwide solid state drive 2008-2012 forecast and analysis. IDC, Technical Report 212736 (2008)
4. Hetzler, S.R.: The storage chasm: Implications for the future of HDD and solid state storage (December 2008), http://www.idema.org
5. TPC-E, http://tpc.org/tpce/default.asp
6. Koltsidas, I., Viglas, S.D.: Flashing up the storage layer. Proc. VLDB Endow. 1(1), 514–525 (2008)
7. Kgil, T., Roberts, D., Mudge, T.: Improving NAND Flash Based Disk Caches. In: Proceedings of the 35th International Symposium on Computer Architecture, June 21-25, pp. 327–338. IEEE Computer Society, Washington (2008)
8. Miller, E.L., Brandt, S.A., Long, D.D.: HeRMES: High-Performance Reliable MRAM-Enabled Storage. In: Proceedings of the Eighth Workshop on Hot Topics in Operating Systems, HOTOS, May 20-22, p. 95. IEEE Computer Society, Washington (2001)
9. Lin, S., Zeinalipour-Yazti, D., Kalogeraki, V., Gunopulos, D., Najjar, W.A.: Efficient indexing data structures for flash-based sensor devices. Trans. Storage 2(4), 468–503 (2006)
10. Moshayedi, M., Wilkison, P.: Enterprise SSDs. Queue 6(4), 32–39 (2008)
11. Narayanan, D., Thereska, E., Donnelly, A., Elnikety, S., Rowstron, A.: Migrating server storage to SSDs: analysis of tradeoffs. In: Proceedings of the 4th ACM European Conference on Computer Systems, EuroSys 2009, Nuremberg, Germany, April 01 - 03, pp. 145–158. ACM, New York (2009)
12. Adaptec MaxIQ^{TM} SSD Cache, http://www.adaptec.com/en-US/products/CloudComputing/MAXIQ/
13. Lee, S.-W., Moon, B.: Design of Flash-based DBMS: an In-Page Logging Approach. In: Proceedings of the ACM SIGMOD, pp. 55–66 (2007)
14. Park, S.-Y., Jung, D., Kang, J.-U., Kim, J.-S., Lee, J.: CFLRU: a Replacement Algorithm for Flash Memory. In: The 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES 2006), pp. 234–241 (October 2006)
15. Poess, M., Nambiar, R.O.: Energy Cost, The Key Challenge of Today's Data Centers: A Power Consumption Analysis of TPC-C Results. In: Proceedings of VLDB (2008)
16. Intel X25 SSD drives, http://www.intel.com/design/flash/nand/extreme/index.htm
17. Kavalanekar, S., Narayanan, D., Sankar, S., Thereska, E., Vaid, K., Worthington, B.: Measuring Database Performance in Online Services: A Trace-Based Approach. In: Nambiar, R., Poess, M. (eds.) TPCTC 2009. LNCS, vol. 5895, pp. 132–145. Springer, Heidelberg (2009)

Benchmarking Adaptive Indexing

Goetz Graefe[2], Stratos Idreos[1], Harumi Kuno[2], and Stefan Manegold[1]

[1] CWI Amsterdam
The Netherlands
first.last@cwi.nl
[2] Hewlett-Packard Laboratories
Palo Alto, CA
first.last@hp.com

Abstract. Ideally, realizing the best physical design for the current and all subsequent workloads would impact neither performance nor storage usage. In reality, workloads and datasets can change dramatically over time and index creation impacts the performance of concurrent user and system activity. We propose a framework that evaluates the key premise of adaptive indexing — a new indexing paradigm where index creation and re-organization take place automatically and incrementally, as a side-effect of query execution. We focus on how the incremental costs and benefits of dynamic reorganization are distributed across the workload's lifetime. We believe measuring the costs and utility of the stages of adaptation are relevant metrics for evaluating new query processing paradigms and comparing them to traditional approaches.

1 Introduction

Consider the task of selecting and constructing indexes for a database that contains hundreds of tables with tens of columns each; a horrendous task if assigned purely to a DB administrator. Figure 1 illustrates the various methods on how to reach an appropriate physical design.

The simplest approach (top row) loads data quickly without indexes, and then does a full table scan for every query. Traditional offline approaches (2nd row) invest the effort to create certain indexes, and then enjoy fast queries on those indexed columns. A third approach is based on online tuning and loads data quickly without indexes. It first observes the workload and then identifies and constructs the most promising indexes [2,12,13]. Unlike all of these methods, *adaptive indexing* (bottom row) creates and refines indexes automatically and incrementally, as a side effect of query execution [6,7,9,10,11].

Each approach is ideal for certain scenarios, depending on how much workload knowledge and idle time is available to invest in preparations, how much storage space and maintenance effort we can afford to spend, and, last but not least, workload composition. For example, if a workload is well-understood and extremely unlikely to change, then it might be most effective to create indexes up-front, as shown in Figure 1(2).

R. Nambiar and M. Poess (Eds.): TPCTC 2010, LNCS 6417, pp. 169–184, 2011.

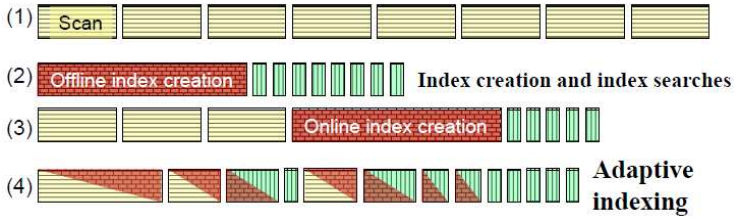

Fig. 1. Four approaches to indexing with regard to query processing

However, one can think of other scenarios with sudden, unpredictable, and radical workload changes. For example, usage of search engines follows current trends on news and human interest. A sudden event somewhere in the world is followed by millions of users searching for the same patterns for a limited amount of time. One cannot anticipate these events before-hand. Any effort to improve performance should have instant results, yet may be useful only during this workload peak, burdening the system afterwards unnecessarily with extra storage and maintenance effort. Adaptive indexing, Figure 1(2), can respond to workload changes automatically, yet without over-investing in short-lived trends.

Contributions. The first two approaches of Figure 1 have been extensively studied in the past, and recently an initial approach for benchmarking online selection (the third approach) has been proposed [14]. In this paper, we set forth a framework for evaluating the new approach of adaptive indexing so that we can properly and systematically compare adaptive indexing techniques as well as identify their benefits over past approaches in dynamic scenarios.

Dynamic Workloads. Adaptive indexing targets dynamic and unpredictable scenarios. For example, in a scientific database, researchers perform exploratory analysis to interpret the data and infer underlying patterns. Workload patterns continuously evolve with their exploration of the data [15]. With new scientific data arriving on a daily basis and with changing search patterns, there is little or no chance to settle for just one workload and create indexes only for that. By the time we have created indexes for one workload pattern, the focus may have already changed. In addition, blindly replicating data in such huge data sets is not appropriate given the already extensive use of storage resources of the ever-expanding data set.

With Knowledge and Time. Traditional approaches for physical design tuning are designed with a drastically different scenario in mind. Perfect up-front workload knowledge is assumed while workloads are assumed to be stable with enough idle time to accommodate traditional physical design. More recent approaches, i.e., soft indexes and online tuning [2,13], go one step further by providing a monitoring step that tries to understand the workload while the system is working and only then create the proper indexes. This deals with situations

where the workload is not known up-front but it also increases the delay of reaching good performance since queries during the monitoring period are not supported by indexes. Such approaches only make sense for scenarios where the time needed to spend in monitoring and the time needed to create the proper physical design are in proportion to the workload span and query performance without indexes is acceptable.

Continuous Physical Adaptation. Adaptive indexing, a very recent development in database architecture technology, addresses the above problems. Instead, of requiring monitoring and preparation steps in order to select and create useful indexes, index creation becomes an integral part of query processing via *adaptive database kernels*. The actual query execution operators and algorithms are responsible for changing the physical design. Essentially, indexes are built selectively, partially and incrementally as a side-effect of query execution. Physical changes do not happen after a query is processed. Instead, they happen while processing the query and are part of query execution.

Adaptive indexing can drastically change the way a database system operates. It also drastically changes the way we should evaluate query processing performance. Current benchmarks largely consider workload knowledge a given, while the index creation overhead is not taken into account as part of the processing costs. However, an adaptive indexing technique and relevant application scenarios need to be evaluated in an entirely different setting considering the complete costs as well as to take into account the various workload phases and how the system performance evolves. This changes the picture dramatically.

A New Benchmark. A recent benchmark proposal formalizes special requirements for online index selection [14]. Unlike established traditional benchmarks, new index structures are intended to be created on-the-fly, so this benchmark takes into account the cost of index creation.

In this paper, we outline a new benchmark specifically targeted for the evaluation of adaptive indexing implementations. As in online tuning, base data is loaded without incurring any time cost for index maintenance before the first query arrives. However, unlike the scenario considered by [14], index creation and maintenance efforts are integrated into query execution actions. Given this incremental, continuous and selective nature of adaptive indexing we need a very different evaluation method than does online index selection.

Breaking the Workload Span. How good or bad an adaptive indexing technique is for a particular workload depends on the overhead that incremental indexing adds to each query and how many queries benefit from the incremental indexing, versus the degree to which that query benefits from the efforts of prior queries. Thus, how an adaptive indexing system compares to a system without index support (in terms of fast loading) or to a system with full index support (in terms of fast queries) depends on how incremental physical design actions are *performed and scheduled* during the workload span. We believe that such new methods are required to evaluate how well query processing techniques serve workloads that are increasingly complex, dynamic, and possibly mixed.

Outline. The rest of this paper is organized as follows. Section 2 briefly discusses related work. Then, Section 3 describes the benchmark in detail while it also provides examples of adaptive indexing behavior and evaluation. Section 4 discusses partial reference implementations of the benchmark, and Section 5 proposes future work. Finally, Section 6 concludes the paper.

2 Related Work and Background

2.1 Classic and Online Indexing

Typically, indexes have been selected and refined by database administrators, possibly using physical design tools, in an offline process based on analyzing a known representative set of the workload [4]. Indexes are then created wholesale. State of the art research also suggests the usage of *alerters* [1,3], i.e., monitoring tools that alert the DBA on when the system should be re-tuned in order to refine a currently suboptimal physical design.

More recently online index creation approaches have been introduced [2,12,13]. They extend the above model for the cases where the workload is not known upfront. They add a monitoring step while the actual workload is being processed and an index is created automatically only once it is believed that it will pay off. Indexes are again created in one go and completely with the difference being that they are created in the background while the workload is actually running.

2.2 Adaptive Indexing

Here, we briefly sketch two adaptive indexing techniques we have recently introduced.

Database Cracking. Database cracking [10] combines some features of both automatic index selection and partial indexes to implement adaptive indexing. As shown in Figure 1(4), it reorganizes data within the query operators, integrating the re-organization effort into query execution. When a column is queried by a predicate for the first time, a new cracker index is initialized. As the column is used in the predicates of further queries, the cracker index is refined by range partitioning until sequentially searching a partition is faster than binary searching in the AVL tree guiding a search to the appropriate partition. The keys in a cracker index are partitioned into disjoint key ranges, but left unsorted within each. Each range query analyzes the cracker index, scans key ranges that fall entirely within the query range, and uses the two end points of the query range to further partition the appropriate two key ranges.

For example, a read-only query on the range "d – p" would crack the keys "y j s c s g m a q k b u e" into three partitions: (1) "c a b e", (2) "j g m k", and (3) "y s q u." If next a new query with range boundaries j and s is processed, the values in partition (1) could be ignored, but partitions (2) and (3) would be further cracked into partitions (2a) "j", (2b) "g m k", (3a)"q s", and (3b)"y u". Subsequent queries continue to partition these key ranges until the structures have been optimized for the current workload.

Updates and their efficient integration into the data structure are covered in [11]. Multi-column indexes to support selections and tuple reconstructions are covered in [9]. Paper [9] also handles storage restrictions via dynamic partial index materialization.

Adaptive Merging. Inspired by database cracking, adaptive merging [6,7] works with block-access devices such as disks, in addition to main memory. The principal goal for designing adaptive merging is to reduce the number of queries required to converge to a fully-optimized index, and the principal mechanism is to employ variable amounts of memory and CPU effort in each query.

While database cracking functions as an incremental quicksort, with each query resulting in at most two partitioning steps, adaptive merging functions as an incremental external merge sort. Under adaptive merging, the first query to use a given column in a predicate produces sorted runs, ideally stored in a partitioned B-tree [5], and subsequent queries upon that same column perform merge steps. Each merge step affects key ranges that are relevant to actual queries, avoiding any effort on all other key ranges. This merge logic executes as a side effect of query execution.

For example an initial read-only query on the range "d – p" might break the keys "y j s c s g m a q k b u e" into equally-sized partitions and then sort them in memory to produce sorted runs : (1) "c j s y", (2) "a g m q", (3) "b e k u". If next a second query with range boundaries j and s is processed, relevant values would be retrieved (via index lookup) and merged out of the runs and into a "final" partition (fully-optimized for the current workload): (1) "c s y", (2) "a g", (3) "b e u", (final) "j k m q". Subsequent queries continue to merge results from the runs until the the "final" partition has been optimized for the current workload.

Hybrids. Currently, we are actively combining ideas from both database cracking and adaptive merging with the goal of combining the strengths of these adaptive indexing techniques so as to better handle dynamic environments.

2.3 Traditional and Online Index Tuning Benchmarks

Traditional benchmarks consider only the query processing costs. More recently, [14] introduces a new benchmark that captures metrics about online indexing techniques. The main distinction of [14] is that it includes the index creation costs as part of processing performance, and is thus suited for evaluating online techniques that create indexes on-the-fly.

Typically, the cumulative cost is considered, i.e., the total time needed to run a workload of Q queries. For an online indexing technique though, [14] includes the cost to run a number of queries without index support as long as the monitoring and decision phase lasts as well as the costs to create the proper indexes and subsequently the cost to run the rest of the workload with index support. The quality of an online technique is based on minimizing this total time as in the case of a classic indexing technique. In online tuning, however, this can be broken down into the individual phases that characterize the costs and benefits of an online technique. For example, one metric is how fast the system

recognizes which indexes to build. The faster this happens, the more queries can benefit from an improved performance.

To evaluate an adaptive indexing technique properly, we must understand both the benefits and also the overhead costs of the incremental index improvements. In adaptive indexing, indexes are built continuously with every query. Each query runs several operators and each operator performs incremental physical changes that improve the structure of indexes. Thus, indexes are not built at once, but rather in numerous steps that are determined by the workload. In fact, an index may never reach a final, fully refined, state if a less-refined state can provide sufficiently good performance for the current workload. In this way, the setting for evaluating adaptive indexing technique is drastically different. We need to carefully identify and evaluate the multiple different stages that index creation goes through until it reaches a stable status. During each stage, the performance characteristics are different. The duration of each of these stages characterizes the quality of an adaptive indexing technique. In the following section, we discuss these concepts in detail.

3 Framework for Adaptive Indexing Benchmarks

As the authors of two different approaches to adaptive indexing [6,7,9,10,11] and long-time admirers of previous self-tuning approaches [1,2,3,12,13] we propose here a framework for benchmarking adaptive indexing systems. One design goal is that the framework should be able to measure the incremental costs and benefits of reorganization actions in terms of how these are distributed along the lifetime of a workload. A second design goal is that this framework should be generic enough that a variety of utility functions and workloads could be used. For example, an implementation of this benchmark could be used to compare two adaptive indexing techniques on the basis of the time needed to execute the workload just as well as on the basis of the energy used. Finally, the framework should support the comparison of the effectiveness of an adaptive indexing technique to any other indexing technique, whether adaptive, online or traditional. For example, given a utility function and a workload, a database practitioner should be able to determine how an adaptive indexing technique compares to a traditional approach in terms of that workload.

We begin by identifying stages that describe how the costs and benefits of adaptive indexing are distributed along the lifetime of a workload. Distinguishing between these stages informs comparisons between different adaptive indexing techniques. Second, we discuss how general design considerations, such as workload composition and query selection, can impact adaptive indexing performance in each of these stages. Finally, we discuss the use of metrics to compare adaptive indexing performance across the lifetime of a workload.

3.1 Stages of Adaptive Indexing

We define stages of an adaptive indexing life-cycle in terms of the overhead that incremental indexing adds to each query, versus the degree to which that query benefits from the efforts of prior queries.

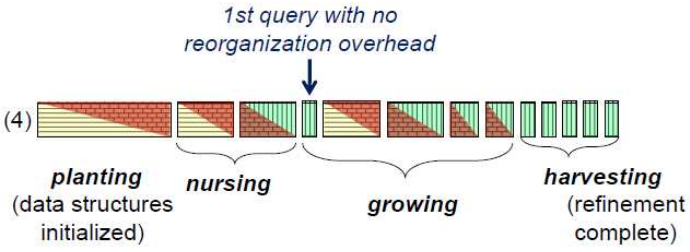

Fig. 2. Adaptive Indexing Stages

As shown in Figure 2, these two metrics help us to identify four stages of adaptive indexing over the lifespan of a workload phase. As starting point, we assume that all data has been loaded into the database system, but no index structures have been created, yet.

Stage 1: Planting. This first stage is characterized by the fact that per-query costs of adaptive indexing exceed those of scan-based query evaluation. The initial step in adaptive indexing is always the extraction of future index entries from the original data collection, e.g., a table stored unsorted and in row format. Even this step can be implemented as side effect of query execution. Subsequently, the index is refined as a side effect of each query. During the planting stage, the expenses for initializing and refining the index exceed the benefits of exploiting the still rudimentary index. Consequently, the total per-query costs are larger than with scan-based query evaluation.

Stage 2: Nursing. With more queries being executed, the index becomes more detailed and complete, yet query execution benefits from the efforts of prior queries. Hence, the expenses for further refining the index decrease while at the same time the benefits of using the improving index increase. Consequently, the per-query costs of adaptive indexing decrease. The point where the per-query costs of adaptive indexing become lower than those of scan-based query evaluation marks the beginning of this second stage. During the nursing stage, the investments of the planting stage start paying-off in terms of per-query costs. However, the cumulative costs over all queries for adaptive indexing still exceed those of scan-based query evaluation.

Stage 3: Growing. As index refinement proceeds, the cumulative benefits of the nursing stage eventually outweigh the cumulative investments during the planting stage. The first query that benefits from the restructuring efforts of previous queries without having to expend any further effort itself beginning of the growing stage, i.e., the stage at which the index structure begins to converge to an optimal state for the current workload phase.

Stage 4: Harvesting. Finally, the index structure is fully optimized and query execution no longer includes side effects. Per-query execution costs reach a minimum. We refer to this final stage as harvesting.

Discussion. The above metrics are drastically different than simply measuring the performance of an a priori fully optimized system or simply considering a one-time index creation online. In adaptive indexing individual queries perform small physical design changes and optimal physical design is reached only after a number of queries have been processed. For adaptive indexing an index is optimal if it allows the current query to be processed in the same time as a fully materialized and fully optimized traditional index. This does not mean though that the adaptive index is completely tuned at this point for the complete data set. It even does not mean that the adaptive index is completely materialized.

Adaptive indexing stages apply to both new non-clustered (secondary, redundant) indexes, as well as to individual key ranges. Applying adaptive indexing to clustered (primary) indexes is more akin to table reorganization rather than index creation. We note that the four stages defined above do not necessarily occur only once per workload, but rather once per index (possibly partial) that a workload phase requires.

While originally defined for adaptive indexing, we can also fit traditional a priori index creation and online index selection/creation into the 4-stages framework. For traditional a priori index creation, the planting stage consists of the actual index creation, and the remainder of the workload moves directly into the harvesting stage. For online index creation, the planting stage covers the initial workload monitoring that leads to the index creation and the index creation itself. After this, the remainder of the workload phase moves directly into the harvesting stage.

3.2 Design Considerations

A benchmark should evaluate the design tradeoffs made by the techniques it evaluates. For example, an online index selection benchmark may test how the allocation of a space budget, the monitoring time period, and the analysis budget impact performance of an index selection technique. In the case of adaptive indexing, because index creation and refinement takes place automatically during the execution of individual queries, there is no monitoring time period, analysis, or even index selection needed. Instead, an adaptive indexing benchmark should test how workload composition and the amount of work performed by query execution side-effects impact each stage of the adaptive indexing process. For a given technique and workload, certain stages might become longer or shorter or even be skipped entirely.

Workload Phases. Because adaptive indexing particularly targets shifting workloads, we model a workload W as a sequence of phases. Each workload phase P comprises a sequence of queries Q and a scheduling discipline S that determines how they will be submitted to the database: $P = (Q, S)$.

Each phase of a workload potentially calls for new index structures and thus passes through the planting, nursing, growing, and harvesting stages. When there is a gradual transition between phases, queries associated with the old phase may be in growing or harvesting stages while queries associated with the new phase

must begin at the planting stage, although it is possible that the preliminary stages of the new phase may be skipped or at least facilitated by work done during a prior phase.

We can model any given indexing mechanism as a transformation function that transforms Phase P's original sequence of queries Q into a new sequence of queries Q' at runtime: $transform(Q, S) = (Q', S)$. Each query $q \in Q$ is transformed individually, depending on its place in the workload.

Utility. Assume there exists a measure of utility $utility(q)$ that applies to the execution of each query and that can also be applied to stages, phases and workloads. For example, one measure might be the time needed for a workload phase to complete: $utility(P) = 1/time(P)$. Other simple measures might be the power used during execution of a query: $utility(q) = 1/power(q)$, or the number of records touched during query execution: $utility(q) = 1/records\_accessed(q)$.

During the planting stage, each transformed query in $q' \in Q'$ has less, or at best equal, utility than its original counterpart. During the nursing stage, some transformed queries have increased utility compared to their original counterparts. At the growing stage, all transformed queries have increased utility dominate those with decreased utility. Finally, during the harvesting stage, the index structure is fully optimized from the perspective of that particular workload phase, and all remaining queries are overhead-free.

Metrics. There are a number of ways to assess the utility of an adaptive indexing mechanism with regard to a given workload. We can assess the overall impact of an adaptive indexing mechanism by comparing the utility of the original and transformed workload. We can compare the overall efficiency of adaptive indexing by comparing the utility of the transformed workload to the utility of a workload with pre-populated indexes.

In addition, because the premise of adaptive indexing is that a workload can reap immediate benefits with low initial investments, we should also consider the cost of the planting and nursing stages, as well as the utility of queries within the nursing and growing stages. To this end, we can measure the aggregate utility per query. Finally, we can consider the speed of convergence (how many queries it takes to reach the harvesting stage).

Experimental Parameters. A number of factors impact how the above metrics are met with regard to a given workload, and that benchmark specifications should consider. The goal of an adaptive indexing benchmark would be to stress an adaptive indexing technique regarding its ability to maintain a fluid and quick transition from one stage to the next. The ideal goal of an adaptive indexing technique is to quickly move through all stages and reach the harvesting stage. The even more crucial part is that it quickly enters the nursing and growing stages so that it can materialize immediate benefits when the workload changes. Thus, critical parameters to study include:

- Varying the range and distribution of keys used in query predicates. Shifting the focus into different key ranges forces an adaptive indexing technique to exit the harvesting stage and return to previous stages. Once the index is again optimized enough or completely for the new key ranges, we again enter the growing and harvesting stage. The smallest the disruption of the stages, the best the adaptive indexing technique is.
- Varying the density of phases per workload rewards strategies that can adapt quickly to a new phase. With workload phases changing rapidly there is less time to spend in adapting so instant reactions and benefits are needed.
- Varying the overlap between workload phases rewards strategies that can leverage effort done during prior phases.
- Varying the number of columns involved in query predicates as well as the tables used in the query workload stresses the ability of an adaptive indexing technique to focus into multiple parts of the data at the same time. It typically extends the length of the stages as it takes more queries and time to improve performance on a given data part. It also stresses the ability of the system to maintain and administer an extended table of contents efficiently.
- Varying the concurrency of queries (the scheduling policy) stresses the ability of an adaptive indexing technique to properly schedule or serialize multiple queries. Ideally, the stages should show the same behavior as if the queries arrive one after the other.
- Varying the percentage of updates in the workload stresses the ability of an adaptive indexing technique to not disturb the stages flow while new data are merged and affect the physical actions performed. At worse an update invalidates all previous actions and leads back to the planting stage. Adaptive indexing techniques though should rely on incremental and differential approaches in order to maintain the stage development.
- Varying the amount of available storage stresses adaptive indexing for its ability to gain and exploit partial knowledge when storage is not enough to accommodate all necessary indexes. It should be able to work on partially materialized indexes and continuously refine them, augment them and reduce them as the workload shifts. Again a powerful adaptive indexing technique is characterized for its ability to quickly go past the planing stage and materialize performance benefits.

4 Partial Reference Implementation

In this section, we illustrate the stages and metrics described in Section 3 using the results of experiments previously published in [9] and [7].

4.1 General Experimental Setup

The ensuing experiments assume no up-front knowledge and, crucially, no idle time to invest in any preparation. Data is loaded up-front in a raw format and queries immediately arrive in a single stream.

The database cracking implementation is built on top of the MonetDB open-source column-store, which resulted in the design of new operators and optimizer rules in the kernel of MonetDB. Experiments were run using a 2.4 GHz Intel Core2 Quad CPU equipped with one 32 KB L1 cache per core, two 4 MB L2 caches, each shared by 2 cores, and 8 GB RAM. The operating system is Fedora 12. The reported experiments for database cracking measure the elapsed execution time for each query processed.

The adaptive merging experiments were done using a simulator capable of configuring experimental parameters such as workspace size, merge fan-in, initial partition size, etc. The metric used is the number of records touched by each query (as opposed to the number of comparisons) which is appropriate for evaluating techniques that target movements in the memory hierarchy.

4.2 Planting, Nursing, and Growing Stages

Our first experiment runs 1000 simple selection queries defined on a 3 attribute table of 10^7 tuples with unique integers randomly located in the table columns. Having a data workload that equally spans across the value domain is again the hardest scenario for adaptive indexing as it offers no flexibility to focus and improve certain areas. The queries are of the following form:

$$\text{select } max(B),max(C) \text{ from R where } v_1 < A < v_2$$

The queries are focused on a particular range of data — we choose v_1 and v_2 such that 9/10 queries request a random range from the first half of the attribute value domain, while only 1/10 queries request a random range from the rest of the domain. All queries request 20% of the tuples.

Figure 3 is based upon Figure 6 of [9], and compares the performance of database cracking (blue), full-sort/traditional indexing (magenta), and scan-only (red) approaches. The utility function is based on elapsed execution time of each query.

The green annotations mark the planting, nursing, and growing stages of database cracking. The nursing stage begins when the first time the cost executing a database cracking query is less than the cost of executing a scan-only query. The growing stage begins with the tenth query — the first that does not incur a cracking action. Note that because the cracking overhead is minimal, in practice, performance at the growing stage eventually matches the full-sort performance. Actually, even in the nursing stage individual query response times are significantly improved over the scan approach and only marginally slower than the presort one.

For the full-sort approach, only the harvesting stage is shown. The presorting cost (planting stage) for the full-sort approach was 3.5 seconds and is not shown on the graph. In other words, this approach assumes perfect knowledge and idle time to prepare. It represents the optimal behavior after paying a hefty cost in the beginning of the query sequence. Since the planting stage fully refines data structures, no nursing or growing stages take place.

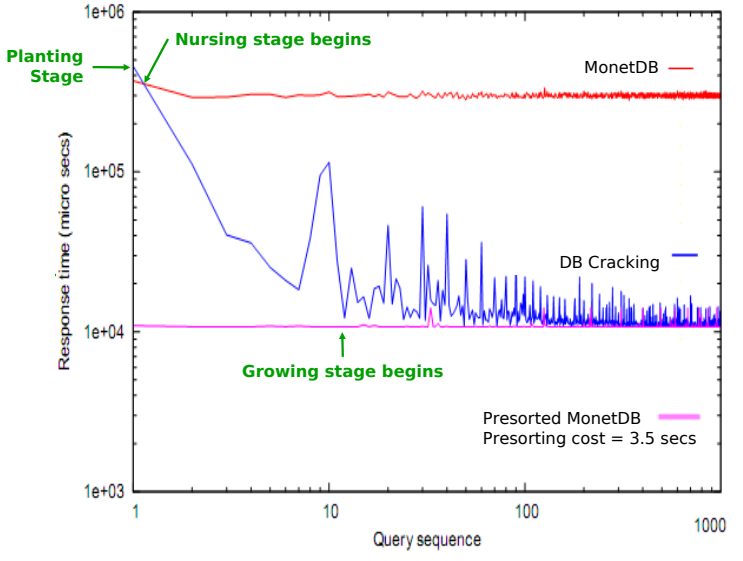

Fig. 3. Adaptive indexing stages illustrated by database cracking

The scan-only approach builds no auxiliary data structures, and thus does not participate in any of the adaptive indexing stages at all. It cannot exploit the fact that the query workload is mainly focused on a small area of the data.

4.3 Shorter Stages

Next we consider the results of an equivalent experiment run using adaptive merging. Adaptive merging is designed with the property of reaching faster the harvesting stage in terms of queries needed. It invests a bit more effort than cracking during the planting stage but less effort than a full sort approach.

This experiment uses a workload of 5,000 queries. Queries are against a random permutation of the integers 0 to 9,999,999. Each query requests a random range of 1 value to 20% of the domain; 10% on average. Initial runs in the partitioned B-tree are created with a workspace of 100,000 records, for 51 initial partitions. The merge fan-in is sufficient to complete all B-tree optimization in a single merge level.

Figure 4 is based upon Figure 9 of [7], and compares the performance of adaptive merging (red), full-sort/traditional indexing (purple), and scan-only (green) approaches. Each data point shows the average of 1% of the workload or 50 queries. Note that in this graph, the utility function is based on the number of records accessed by each query.

The scan and presort options show the same behavior as in the previous experiment. Adaptive indexing though shows a different behavior with the stages being much shorter. The red annotations mark the nursing, and harvesting stages

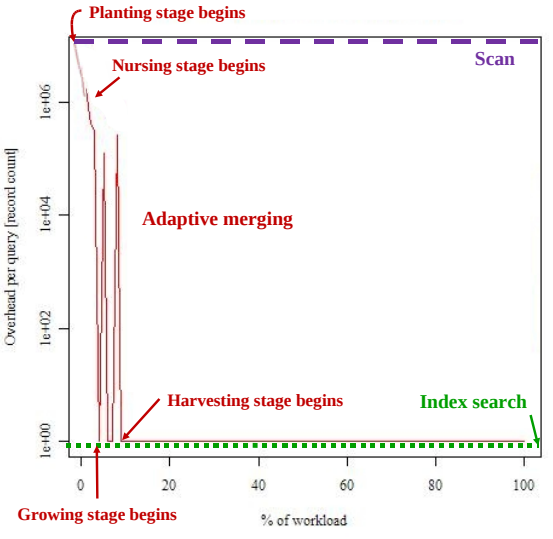

Fig. 4. Adaptive merging query sequence: shorter stages

of adaptive merging. Given the active nature of adaptive merging the harvesting stage begins very fast leaving a fully optimized B-tree after less than 50 queries.

4.4 Multiple Workload Phases, Including Updates

The experiments described above each address only a single phase. Our next two experiments illustrates adaptive indexing in the context of a workload with multiple phases. We first consider a workload consisting of ten phases representing drifting range queries with a drifting focus, as executed by adaptive merging. We next consider a workload consisting of update and read-only query phases, as executed by database cracking.

Drifting Query Focus. One of the design goals of adaptive indexing is to focus index optimization effort on key ranges that occur in actual queries. If the focus of query activity changes, additional key ranges are optimized in the index as appropriate. In this workload, 10^7 records with unique key values are searched by 500 queries in five phases that shift focus from the lowest key range in the index to the highest key range.

Figure 5 is based upon Figure 18 of [8], illustrates the overhead per query as the workload passes through the phases. As in the previous adaptive merging experiment, the utility function is based on the number of records accessed by each query. Because the data accessed by the various phases does not overlap, each new phase must pass through new nursing, growing, and harvesting stages.

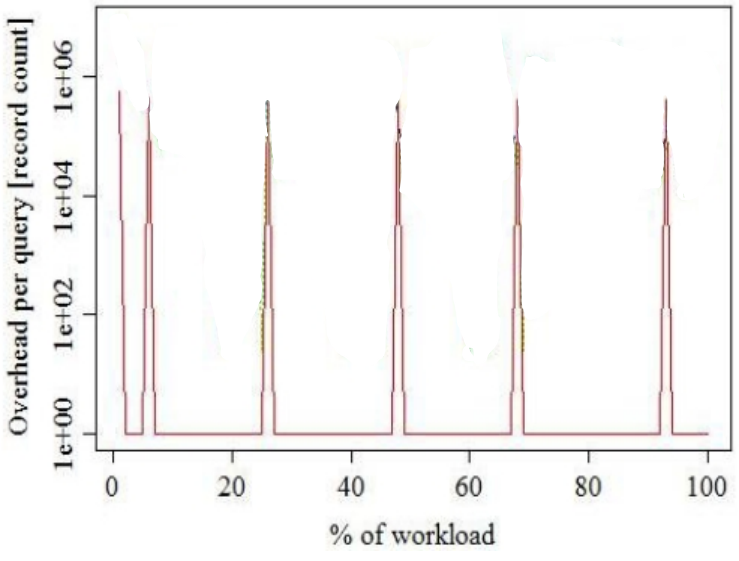

Fig. 5. Workload with five phases of query focus illustrated by adaptive merging

Mixture of Updates and Read-Only Queries. Next we consider a workload that contains a random mixture of update and query phases. Naturally, updates require some auxiliary work which pose an overhead that may eventually disturb the normal flow of adaptive indexing stages.

Two scenarios are considered here, (a) the high frequency low volume scenario (HFLV); every 10 queries we get 10 random updates and (b) the low frequency high volume scenario (LFHV); every 10^3 queries we get 10^3 random updates. Random queries are used in the same form as for the first cracking experiment. Using completely random queries represents the most challenging workload for an adaptive technique as there is basically no pattern to adapt to. In other words, using a random workload will result in the longest possible stages in the adaptation procedure.

Figure 6 is based upon Figure 7 of [9], and shows that cracking maintains high performance and a self-organizing behavior through the whole sequence of queries and updates. Peaks and bumps occur frequently disturbing momentarily the current stage every time.

The power of an adaptive indexing technique is how well it can absorb these peaks. To achieve this an adaptive indexing needs to rely on adaptive and incremental methods for handling updates. In this case, updates are handled during query processing as part of the incremental physical design changes, i.e., the actual query processing operators in the DB kernel are responsible for on-the-fly merging the necessary updates.

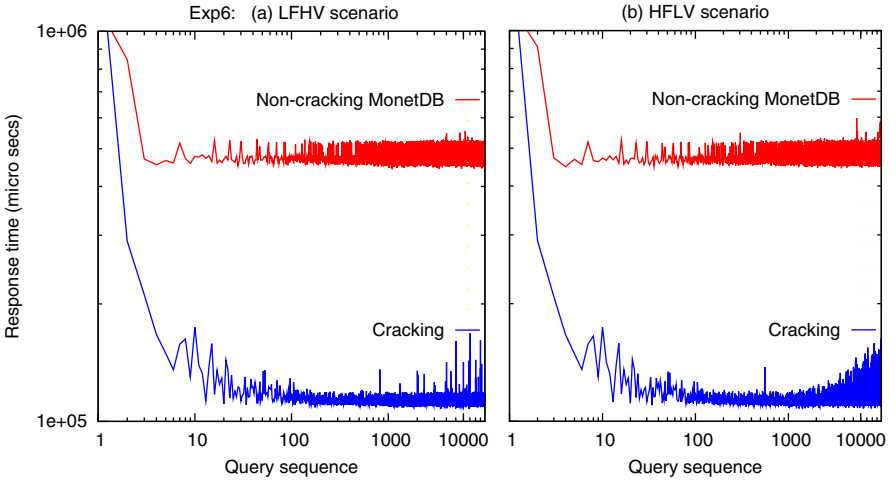

Fig. 6. Effect of updates

5 Outlook

Adaptive indexing techniques (database cracking, adaptive merging, and hybrids) can be combined with automatic index tuning in multiple ways. A tuning tool might prevent certain indexes (e.g., due to excessive anticipated update costs) or it might encourage certain indexes (e.g., for immediate creation as side effect during the first appropriate query execution). Alternatively, the tuning tool might observe activity and pursue index optimization proactively without waiting for queries and their side effects. It might perform only some initial steps of adaptive index creation and optimization (e.g., extraction of future index entries and run generation, but not merging) or it might finish partially optimized indexes (e.g., sort small partitions left by database cracking). In addition, a tuning tool could set resource-usage based policies that limit adaptive indexing during query execution (e.g., based on memory allocation during run generation or merging). We intend to explore in our future research some or all of these combinations of adaptive techniques with traditional index tuning techniques. Benchmarks that measure and compare costs and benefits of such hybrid techniques will increase our understanding and guide database developers when choosing techniques to implement and when guiding the application developers.

6 Summary

In this paper, we have laid out the first framework for benchmarking adaptive indexing techniques. We have described the problem of adaptive indexing, discussed characteristics that differentiate adaptive indexing approaches from alternatives, and proposed a framework for comparing these characteristics. Unlike traditional indexing techniques, adaptive indexing distributes the effort of

indexing incrementally across the workload as a side effect of query execution. Each phase of a workload goes through distinct stages of the adaptive indexing life-cycle in terms of the overhead that incremental indexing adds to each query, versus the degree to which that query benefits from the efforts of prior queries. An adaptive indexing benchmark for dynamic database scenarios must take both workload phases and adaptive indexing stages into account, including stressing the system's ability to maintain a rapid and fluid transition from one stage to the other. For the sake of illustration, we described our partial reference implementation of a benchmark instance using this framework.

Adaptive indexing and the ways to evaluate it represent a completely new paradigm. We believe the new evaluation methods presented here can also be exploited by existing offline and online techniques to improve performance in dynamic scenarios.

References

1. Bruno, N., Chaudhuri, S.: To tune or not to tune? a lightweight physical design alerter. In: VLDB (2006)
2. Bruno, N., Chaudhuri, S.: An online approach to physical design tuning. In: ICDE (2007)
3. Bruno, N., Chaudhuri, S.: Physical design refinement: the 'merge-reduce' approach. In: ACM TODS (2007)
4. Chaudhuri, S., Narasayya, V.R.: Self-tuning database systems: A decade of progress. In: VLDB (2007)
5. Graefe, G.: Sorting and indexing with partitioned b-trees. In: CIDR (2003)
6. Graefe, G., Kuno, H.: Adaptive indexing for relational keys. In: SMDB (2010)
7. Graefe, G., Kuno, H.: Self-selecting, self-tuning, incrementally optimized indexes. In: EDBT (2010)
8. Graefe, G., Kuno, H.: Two adaptive indexing techniques: improvements and performance evaluation. In: HPL Technical Report (2010)
9. Idreos, S., Kersten, M., Manegold, S.: Self-organizing tuple reconstruction in column stores. In: SIGMOD (2009)
10. Idreos, S., Kersten, M.L., Manegold, S.: Database cracking. In: CIDR (2007)
11. Idreos, S., Kersten, M.L., Manegold, S.: Updating a cracked database. In: SIGMOD (2007)
12. Lühring, M., Sattler, K.-U., Schmidt, K., Schallehn, E.: Autonomous management of soft indexes. In: SMDB (2007)
13. Schnaitter, K., Abiteboul, S., Milo, T., Polyzotis, N.: COLT: continuous on-line tuning. In: SIGMOD (2006)
14. Schnaitter, K., Polyzotis, N.: A benchmark for online index selection. In: ICDE (2009)
15. Tukey, J.W.: Exploratory Data Analysis. Addison-Wesley, Reading (1977)

XWeB: The XML Warehouse Benchmark

Hadj Mahboubi[1] and Jérôme Darmont[2]

[1] CEMAGREF Clermont-Ferrand
24 avenue des Landais, BP 50085, 63172 Aubière Cedex, France
hadj.mahboubi@cemagref.fr
http://eric.univ-lyon2.fr/~hmahboubi/
[2] Université de Lyon (ERIC Lyon 2)
5 avenue Pierre Mendès-France, 69676 Bron Cedex, France
jerome.darmont@univ-lyon2.fr
http://eric.univ-lyon2.fr/~jdarmont/

Abstract. With the emergence of XML as a standard for representing business data, new decision support applications are being developed. These XML data warehouses aim at supporting On-Line Analytical Processing (OLAP) operations that manipulate irregular XML data. To ensure feasibility of these new tools, important performance issues must be addressed. Performance is customarily assessed with the help of benchmarks. However, decision support benchmarks do not currently support XML features. In this paper, we introduce the XML Warehouse Benchmark (XWeB), which aims at filling this gap. XWeB derives from the relational decision support benchmark TPC-H. It is mainly composed of a test data warehouse that is based on a unified reference model for XML warehouses and that features XML-specific structures, and its associate XQuery decision support workload. XWeB's usage is illustrated by experiments on several XML database management systems.

Keywords: benchmark, XML data warehouse, OLAP, TPC-H.

1 Introduction

With the increasing volume of XML data available, and XML now being a standard for representing complex business data [2], XML data sources that are pertinent for decision support are ever more numerous. However, XML data bear irregular structures (e.g., optional and/or diversely ordered elements, ragged hierarchies, etc.) that would be intricate to handle in a relational Database Management System (DBMS). Therefore, many efforts toward XML data warehousing have been achieved [14,17,29], as well as efforts for extending the XQuery language with On-Line Analytical Processing (OLAP) capabilities [9,12,26].

XML-native DBMSs supporting XQuery should naturally form the basic storage component of XML warehouses. However, they currently present relatively poor performances when dealing with the large data volumes and complex analytical queries that are typical in data warehouses, and are thus challenged by relational, XML-compatible DBMSs. A tremendous amount of research is

R. Nambiar and M. Poess (Eds.): TPCTC 2010, LNCS 6417, pp. 185–203, 2011.

currently in progress to help them become a credible alternative, though. Since performance is a critical issue in this context, its assessment is primordial.

Database performance is customarily evaluated experimentally with the help of benchmarks. However, existing decision support benchmarks [7,15,16,24] do not support XML features, while XML benchmarks [4,5,20,28] target transactional applications and are ill-suited to evaluate the performances of decision-oriented applications. Their database schemas do not bear the multidimensional structure that is typical in data warehouses (i.e., star schemas and derivatives bearing facts described by dimensions [11]); and their workloads do not feature typical, OLAP-like analytic queries.

Therefore, we present in this paper the first (to the best of our knowledge) XML decision support benchmark. Our objective is to propose a test XML data warehouse and its associate XQuery decision support workload, for performance evaluation purposes. The XML Warehouse Benchmark (XWeB) is based on a unified reference model for XML data warehouses [14]. An early version of XWeB [13] was derived from the standard relational decision support benchmark TPC-H [25]. In addition, XWeB's warehouse model has now been complemented with XML-specific irregular structures, and its workload has been both adapted in consequence and expanded.

The remainder of this paper is organized as follows. In Section 2, we present and discuss related work regarding relational decision support and XML benchmarks. In Section 3, we recall the XML data warehouse model XWeB is based on. In Section 4, we provide the full specifications of XWeB. In Section 5, we illustrate our benchmark's usage by experimenting on several XML DBMSs. We finally conclude this paper and provide future research directions in Section 6.

2 Related Work

2.1 Relational Decision Support Benchmarks

The OLAP APB-1 benchmark has been very popular in the late nineties [15]. Issued by the OLAP Council, a now inactive organization founded by four OLAP solution vendors, APB-1's data warehouse schema is structured around *Sale* facts and four dimensions: *Customer*, *Product*, *Channel* and *Time*. Its workload of ten queries aims at sale forecasting. Although APB-1 is simple to understand and use, it proves limited, since it is not "differentiated to reflect the hurdles that are specific to different industries and functions" [22].

Henceforth, the Transaction Processing Performance Council (TPC) defines standard benchmarks and publishes objective and verifiable performance evaluations to the industry. The TPC currently supports one decision support benchmark: TPC-H [25]. TPC-H's database is a classical *product-order-supplier* model. Its workload is constituted of twenty-two SQL-92, parameterized, decision support queries and two refreshing functions that insert tuples into and delete tuples from the database, respectively. Query parameters are randomly instantiated following a uniform law. Three primary metrics are used in TPC-H. They describe

performance in terms of power, throughput, and a combination of these two criteria. Power and throughput are the geometric and arithmetic mean values of database size divided by workload execution time, respectively.

Although decision-oriented, TPC-H's database schema is not a typical star-like data warehouse schema. Moreover, its workload does not include any explicit OLAP query. The TPC-DS benchmark, which is currently in its latest stages of development, fixes this up [24]. TPC-DS' schema represents the decision support functions of a retailer under the form of a constellation schema with several fact tables and shared dimensions. TPC-DS' workload is constituted of four classes of queries: reporting queries, ad-hoc decision support queries, interactive OLAP queries, and extraction queries. SQL-99 query templates help randomly generate a set of about five hundred queries, following non-uniform distributions. The warehouse maintenance process includes a full Extract, Transform and Load (ETL) phase, and handles dimensions with respect to their nature (non-static dimensions scale up while static dimensions are updated). One primary throughput metric is proposed in TPC-DS to take both query execution and the maintenance phase into account.

More recently, the Star Schema Benchmark (SSB) has been proposed as a simpler alternative to TPC-DS [16]. As our early version of XWeB [13], it is based on TPC-H's database remodeled as a star schema. It is basically architectured around an *order* fact table merged from two TPC-H tables. But more interestingly, SSB features a query workload that provides both functional and selectivity coverages.

As in all TPC benchmarks, scaling in TPC-H, TPC-DS and SSB is achieved through a scale factor SF that helps define database size (from 1 GB to 100 TB). Both database schema and workload are fixed. The number of generated queries in TPC-DS also directly depends on SF. TPC standard benchmarks aim at comparing the performances of different systems in the same experimental conditions, and are intentionally not very tunable. By contrast, the Data Warehouse Engineering Benchmark (DWEB) helps generate various ad-hoc synthetic data warehouses (modeled as star, snowflake, or constellation schemas) and workloads that include typical OLAP queries [7]. DWEB targets data warehouse designers and allows testing the effect of design choices or optimization techniques in various experimental conditions. Thus, it may be viewed more like a benchmark generator than an actual, single benchmark. DWEB's main drawback is that its complete set of parameters makes it somewhat difficult to master.

Finally, to be complete, TPC-H and TPC-DS have recently be judged insufficient for ETL purposes [21] and specific benchmarks for ETL workflows are announced [21,27].

2.2 XML Benchmarks

XML benchmarks may be subdivided into two families. On one hand, micro-benchmarks, such as the Michigan Benchmark (so-named in reference to the relational Wisconsin Benchmark developed in the eighties) [19] and MemBeR [1], help XML documents storage solution designers isolate critical issues to optimize.

More precisely, micro-benchmarks aim at assessing the individual performances of basic operations such as projection, selection, join and aggregation. These low-level benchmarks are obviously too specialized for decision support application evaluation, which requires testing complex queries at a more global level.

On the other hand, application benchmarks help users compare the global performances of XML-native or compatible DBMSs, and more particularly of their query processor. For instance, X-Mach1 [4], XMark [20], XOO7 (an extension of the object-oriented benchmark OO7) [5] and XBench [28] are application benchmarks. Each implements a mixed XML database that is both data-oriented (structured data) and document-oriented (in general, random texts built from a dictionary). However, except for XBench that proposes a true mixed database, their orientation is either more particularly focused on data (XMark, XOO7) or documents (X-Mach1).

These benchmarks also differ in: the fixed or flexible nature of the XML schema (one or several Document Type Definitions – DTDs – or XML Schemas); the number of XML documents used to model the database at the physical level (one or several); the inclusion or not of update operations in the workload. We can also underline that only XBench helps evaluate all the functionalities offered by the XQuery language. Unfortunately, none of these benchmarks exhibit any decision support feature. This is why relational benchmarks presented in Section 2.1 are more useful to us in a first step.

3 Reference XML Warehouse Model

Existing XML data warehouse architectures more or less converge toward a unified model. They mostly differ in the way dimensions are handled and the number of XML documents that are used to store facts and dimensions. Searching for the best compromise in terms of query performance and modeling power, we proposed a unified model [14] that we reuse in XWeB. As XCube [10], our reference XML warehouse is composed of three types of XML documents at the physical level: document *dw-model.xml* defines the multidimensional structure of the warehouse (metadata); each $facts_f.xml$ document stores information related to set of facts f (several fact documents allow constellation schemas); each $dimension_d.xml$ document stores a given dimension d's member values for any hierarchical level.

More precisely, *dw-model.xml*'s structure (Figure 1) bears two types of nodes: *dimension* and *FactDoc* nodes. A *dimension* node defines one dimension, its possible hierarchical levels (*Level* elements) and attributes (including types), as well as the path to the corresponding $dimension_d.xml$ document. A *FactDoc* element defines a fact, i.e., its measures, references to the corresponding dimensions, and the path to the corresponding $facts_f.xml$ document. The $facts_f.xml$ documents' structure (Figure 2(a)) is composed of *fact* subelements that each instantiate a fact, i.e., measure values and dimension references. These identifier-based references support the fact-to-dimension relationships.

Finally, the $dimension_d.xml$ documents' structure (Figure 2(b)) is composed of *Level* nodes. Each of them defines a hierarchy level composed of *instance*

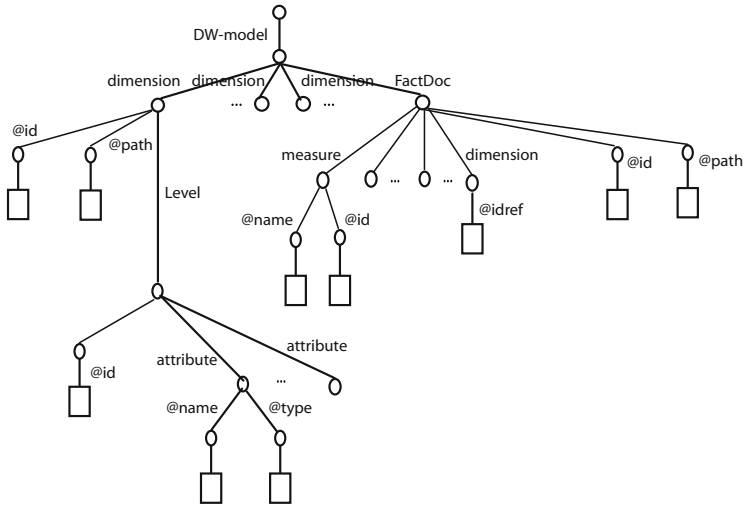

Fig. 1. *dw-model.xml* graph structure

nodes. An *instance* defines the member attributes of a hierarchy level as well as their values.

4 XWeB Specifications

4.1 Principle

XWeB derives from TPC-H, modified in a number of ways explained in the following sections, for three reasons. First, we acknowledge the importance of TPC benchmarks' standard status. Hence, our goal is to have XWeB inherit from TPC-H's wide acceptance and usage (whereas TPC-DS is still under development). Second, from our experience in designing the DWEB relational data warehouse benchmark, we learned that Gray's simplicity criterion for a good benchmark [8] is primordial. This is again why we preferred TPC-H, which is much simpler than TPC-DS or DWEB. Third, from a sheer practical point of view, we also selected TPC-H to benefit from its data generator, `dbgen`, a feature that does not exist in TPC-DS yet.

The main components in a benchmark are its database and workload models. XWeB's are described in Sections 4.2 and 4.3, respectively. In a first step, we do not propose to include ETL features in XWeB, although XQuery has been complemented with update queries recently [6]. ETL is indeed a complex process that presumably requires dedicated benchmarks [21]. Moreover, the following specifications already provide a raw loading evaluation framework. The XWeB warehouse is indeed a set of XML documents that must be loaded into an XML DBMS, an operation that can be timed.

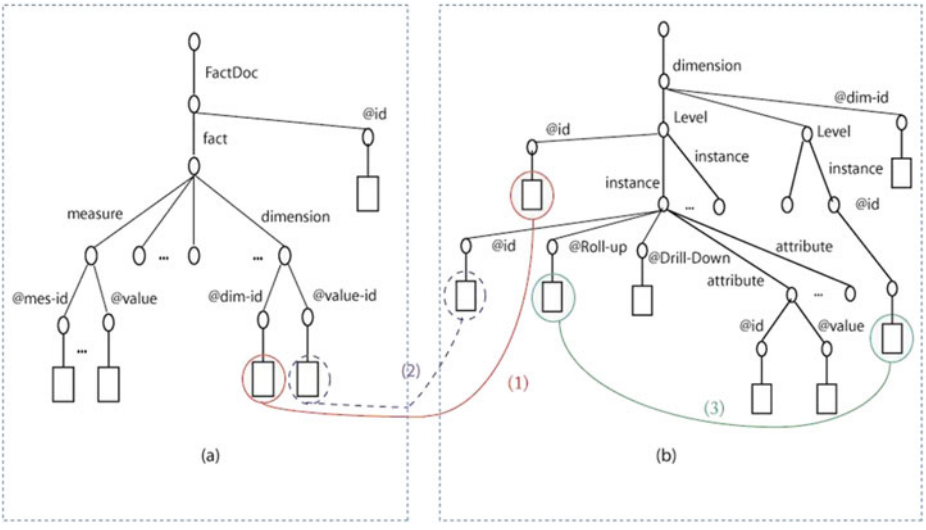

Fig. 2. $facts_f.xml$ (a) and $dimension_d.xml$ (b) graph structures

4.2 Database Model

Schema. At the conceptual level, like O'Neil et al. in SSB, we remodel TPC-H's database schema as an explicit multidimensional (snowflake) schema (Figure 3), where *Sale* facts are described by the *Part/Category*, *Customer/Nation/Region*, *Supplier/Nation/Region* and *Day/Month/Year* dimensions.

The *Part/Category* hierarchy, which is not present in TPC-H, is of particular interest. It is indeed both non-strict and non-covering [23]. Beyer et al. would term it *ragged* [2]. We prefer the term *complex* since ragged hierarchy has different meanings in the literature; e.g., Rizzi defines it as non-covering only [18]. More precisely, in our context, non-strictness means relationships between parts and categories, and between categories themselves, are many-to-many. Non-coveringness means parts and subcategories may roll up to categories at any higher granularity level, i.e., skipping one or more intermediary granularity levels. Complex hierarchies do exist in the real world, are easy to implement in XML, whereas they would be intricate to handle in a relational system [2].

At the logical level, the UML class diagram from Figure 3 translates into an instance of *dw-model.xml* (Figure 4). Attributes (fact measures and dimension members) are not mentioned in Figure 4 for brevity, but they are present in the actual document.

Finally, at the physical level, fact and dimension instances are stored in a set of XML documents, namely $facts_1.xml$ = *f_sale.xml*, $dimension_1$ = *d_date.xml*, $dimension_2$ = *d_part.xml*, $dimension_3$ = *d_customer.xml* and $dimension_4$ = *d_supplier.xml*. To introduce further XML-specific features in XWeB, *f_sale.xml*'s DTD allows missing dimension references and measures, as

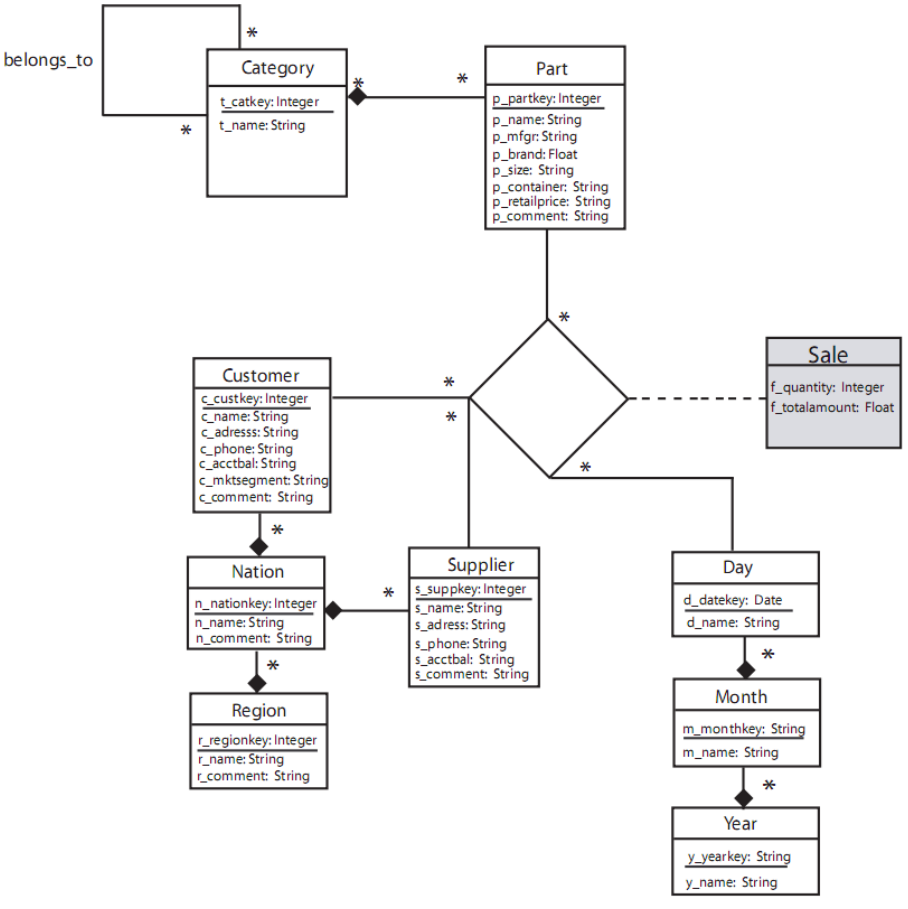

Fig. 3. XWeB warehouse's conceptual schema

well as any order in fact subelements. Our aim here is to introduce a measure of "dirty data" in the benchmark.

Parameterization. XWeB's main parameters basically help control data warehouse size. Size (S) depends on two parameters: the scale factor (SF) inherited from TPC-H, and density D. When $D = 1$, all possible combinations of dimension references are present in the fact document (Cartesian product), which is very rare in real-life data warehouses. When D decreases, we progressively eliminate some of these combinations. D actually helps control the overall size of facts independently from the size of dimensions.

S can be estimated as follows: $S = S_{dimensions} + S_{facts}$, where $S_{dimensions}$ is the size of dimensions, which does not change when SF is fixed, and S_{facts} is the size of facts, which depends on D. $S_{dimensions} = \sum_{d \in \mathcal{D}} |d|_{SF} \times nodesize(d)$ and

```
<?xml version="1.0" encoding="UTF-8"?>
<xweb-dw-model>
  <fact id="Sale" path="f_sale.xml"/>
  <dimension id="Date" path="d_date.xml">
    <level id="Day" rollup="Month" drilldown=""/>
    <level id="Month" rollup="Year" drilldown="Day"/>
    <level id="Year" rollup="" drilldown="Month"/>
  </dimension>
  <dimension id="PartDim" path="d_part.xml"/>
    <level id="Part" rollup="Category" drilldown=""/>
    <level id="Category" rollup="Category" drilldown="Part Category"/>
  </dimension>
  <dimension id="CustomerDim" path="d_customer.xml">
    <level id="Customer" rollup="C_Nation" drilldown=""/>
    <level id="C_Nation" rollup="C_Region" drilldown="Customer"/>
    <level id="C_Region" rollup="" drilldown="C_Nation"/>
  </dimension>
  <dimension id="SupplierDim" path="d_supplier.xml">
    <level id="Supplier" rollup="S_Nation" drilldown=""/>
    <level id="S_Nation" rollup="S_Region" drilldown="Supplier"/>
    <level id="S_Region" rollup="" drilldown="S_Nation"/>
  </dimension>
</xweb-dw-model>
```

Fig. 4. XWeB warehouse's logical schema

$S_{facts} = \prod_{d \in \mathcal{D}} |h_1^d|_{SF} \times D \times fact\_size$, where $\mathcal{D}$ is the set of dimensions, $|d|_{SF}$ the total size of dimension d (i.e., all hierarchy levels included) w.r.t. SF, $|h_1^d|_{SF}$ the size of the coarsest hierarchy level in dimension d w.r.t. SF, $nodesize(d)$ the average node size in $dimension_d.xml$, and $fact\_size$ the average fact element size. For example, when $SF = 1$ and $D = 1$, with node sizes all equal to 220 bytes, the size of $f\_sale.xml$ is 2065 GB. Eventually, two additional parameters control the probability of missing values (P_m) and element reordering (P_o) in facts, respectively.

Schema Instantiation. The schema instantiation process is achieved in two steps: first, we build dimension XML documents, and then the fact document. Dimension data are obtained from `dbgen` as flat files. Their size is tuned by SF. Dimension data are then matched to the *dw-model.xml* document, which contains dimension specifications, hierarchical levels and attribute names, to output the set of $dimension_d.xml$ ($d \in \mathcal{D}$) documents. *d_part.xml*, which features a complex hierarchy, is a particular case that we focus on.

Algorithm from Figure 5 describes how categories are assigned to parts from *d_part.xml*. First, category names are taken from TPC-H and organized in three arbitrary levels in the *cat* table. Moreover, categories are interrelated through

rollup and drill-down relationships to form a non-strict hierarchy. For example, level-2 category BRUSHED rolls up to level-1 categories NICKEL and STEEL, and drills down to level-3 categories ECONOMY, STANDARD and SMALL. The whole hierarchy extension is available on-line (Section 6). Then, to achieve non-coveringness, we assign to each part p several categories at any level. $p.catset$ denotes the set of categories assigned to p. Each "root" category (numbering from 1 to 3) is selected from a random level lvl. Then, subcategories may be (randomly) selected from subsequent levels. Non-coveringness is achieved when initial level is lower than 3 and there is no subcategory. $ncat$ and $nsubcat$ refer to category and subcategory numbers, respectively. $cand$ denotes a candidate category or subcategory. $|cat[i]|$ is the number of elements in table cat's i^{th} level.

```
cat := [[BRASS, COPPER, NICKEL, STEEL, TIN],                  // level 1
        [ANODIZED, BRUSHED, BURNISHED, PLATED, POLISHED],     // level 2
        [ECONOMY, LARGE, MEDIUM, PROMO, SMALL, STANDARD]]     // level 3
FOR ALL p IN d_part DO
   p.catset := EMPTY_SET
   ncat := RANDOM(1, 3)
   FOR i := 1 TO ncat DO
      lvl := RANDOM(1, 3)
      REPEAT
         cand := cat[lvl, RANDOM(1, |cat[lvl]|)]
      UNTIL cand NOT IN p.catset
      p.catset := p.catset UNION cand
      nsubcat := RANDOM(0, 3 - lvl)
      FOR j := 1 TO nsubcat DO
         cand := cat[lvl + j, RANDOM(1, |cat[lvl + j]|)]
         IF cand NOT IN p.catset THEN
            p.catset := p.catset UNION cand
         END IF
      END FOR
   END FOR
END FOR
```

Fig. 5. Part category selection algorithm

Facts are generated randomly with respect to the algorithm from Figure 6. The number of facts depends on D, and data dirtiness on P_m and P_o (Section 4.2). D, P_m and P_o are actually used as Bernouilli parameters. val is a transient table that stores dimension references and measure values, to allow them to be nullified and/or reordered without altering loop index values. The `SKEWED_RANDOM()` function helps generate "hot" and "cold" values for measures $Quantity$ and $TotalAmount$, which influences range queries. Finally, the `SWITCH()` function randomly reorders a set of values.

```
FOR ALL c IN d_customer DO
   FOR ALL p IN d_part DO
      FOR ALL s IN d_supplier DO
         FOR ALL d IN d_date DO
            IF RANDOM(0, 1) <= D THEN
               // Measure random generation
               Quantity := SKEWED_RANDOM(1, 10000)
               TotalAmount := Quantity * p.p_retailprice
               // Missing values management
               val[1] := c.c_custkey; val[2] := p.p_partkey
               val[3] := s.s_suppkey; val[4] := d.d_datekey
               val[5] := Quantity; val[6] := TotalAmount
               FOR i := 1 TO 6 DO
                  IF RANDOM(0, 1) <= Pm THEN
                     val[i] := NULL
                  END IF
               END FOR
               // Dimension reordering
               IF RANDOM(0, 1) <= Po THEN
                  SWITCH(val)
               END IF
               WRITE(val) // Append current fact into f_sale.xml
            END IF
         END FOR
      END FOR
   END FOR
END FOR
```

Fig. 6. Fact generation algorithm

4.3 Workload Model

Workload Queries and Parameterization. The XQuery language [3] allows formulating decision support queries, unlike simpler languages such as XPath. Complex queries, including aggregation operations and join queries over multiple documents, can indeed be expressed with the FLWOR syntax. However, we are aware that some analytic queries are difficult to express and execute efficiently with XQuery, which does not include an explicit grouping construct comparable to the `GROUP BY` clause in SQL [2]. Moreover, though grouping queries are possible in XQuery, there are many issues with the results [2]. We nonetheless select XQuery for expressing XWeB's workload due to its standard status. Furthermore, introducing difficult queries in the workload aims to challenge XML DBMS query engines.

Although we do take inspiration from TPC-H and SSB, our particular XML warehouse schema leads us to propose yet another query workload. It is currently composed of twenty decision support queries labeled Q01 to Q20 that basically are typical aggregation queries for decision support. Though we aim

to provide the best functional and selectivity coverages with this workload, we lack experimental feedback, thus it is likely to evolve in the future. Workload specification is provided in Table 1. Queries are presented in natural language for space constraints, but their complete XQuery formulation is available on-line (Section 6).

XWeB's workload is roughly structured in increasing order of query complexity, starting with simple aggregation, then introducing join operations, then OLAP-like queries such as near-cube (with superaggregates missing) calculation, drill-downs (e.g., Q06 drills from Q05's *Month* down to *Day* granularity) and rollups (e.g., Q09 rolls from Q08's *Customer* up to *Nation* granularity), while increasing the number of dimensions involved. The last queries exploit the *Part/Category* complex hierarchy. We also vary the type of restrictions (by-value and range queries), the aggregation function used, and the ordering applied to queries. Ordering labeled by $^{-1}$ indicates a descending order (default being ascending). Finally, note that Q20, though apparently identical to Q19, is a further rollup along the *Category* complex hierarchy. Actually, Q19 rolls up from Q18's product level to the category level, and then Q20 rolls up to the "supercategory" level, with supercategories being categories themselves.

Moreover, workload queries are subdivided into five categories: simple reporting (i.e., non-grouping) queries; 1, 2, and 3-dimension cubes; and complex hierarchy cubes. We indeed notice in our experiments (Section 5) that complex queries are diversely handled by XML DBMSs: some systems have very long response times, and even cannot answer. Subdividing the workload into blocks allows us to adjust workload complexity, by introducing boolean execution parameters (RE, $1D$, $2D$, $3D$ and CH, respectively) that define whether a particular block of queries must be executed or not when running the benchmark (see below).

Execution Protocol and Performance Metrics. Still with TPC-H as a model, we adapt its execution protocol along two axes. First, since XWeB does not currently feature update operations (Section 4.1), the performance test can be simplified to executing the query workload. Second, as in DWEB, we allow warm runs to be performed several times (parameter $NRUN$) instead of just once, to allow averaging results and flattening the effects of any unexpected outside event. Thus, the overall execution protocol may be summarized as follows:

1. **load test:** load the XML warehouse into an XML DBMS;
2. **performance test:**
   (a) *cold run* executed once (to fill in buffers), w.r.t. parameters RE, $1D$, $2D$, $3D$ and CH;
   (b) *warm run* executed $NRUN$ times, still w.r.t. workload parameters.

The only performance metric in XWeB is currently *response time*, as in SSB and DWEB. Load test, cold run and warm runs are timed separately. Global, average, minimum and maximum execution times are also computed, as well standard deviation. This kind of atomic approach for assessing performance allows to

Table 1. XWeB workload specification

Group	Query	Specification	Restriction	Ordering
Reporting	Q01	Min, Max, Sum, Avg of *f_quantity* and *f_totalamount*		
	Q02	*f_quantity* for each *p_partkey*	$p\_retailprice \leq 1000$	*p_retailprice*
	Q03	Sum of *f_totalamount*	$n\_name =$ "*FRANCE*"	
1D cube	Q04	Sum of *f_quantity* per *p_partkey*	$p\_retailprice > 1500$	$p\_retailprice^{-1}$
	Q05	Sum of *f_quantity* and *f_totalamount* per *m_monthname*	Quarter($m\_monthkey$) = 1	*m_monthname*
	Q06	Sum of *f_quantity* and *f_totalamount* per *d_dayname*	Quarter($m\_monthkey$) = 1	*d_dayname*
	Q07	Avg of *f_quantity* and *f_totalamount* per *r_name*	$r\_name =$ "*AMERICA*"	
2D cube	Q08	Sum of *f_quantity* and *f_totalamount* per *c_name* and *p_name*	$p\_brand =$ "*Brand*#25"	*c_name*, *p_name*
	Q09	Sum of *f_quantity* and *f_totalamount* per *n_name* and *p_name*	$p\_brand =$ "*Brand*#25"	*n_name*, *p_name*
	Q10	Sum of *f_quantity* and *f_totalamount* per *r_name* and *p_name*	$p\_brand =$ "*Brand*#25"	*r_name*, *p_name*
	Q11	Max of *f_quantity* and *f_totalamount* per *s_name* and *p_name*	$s\_acctbal < 0$	*s_name*, *p_name*
3D cube	Q12	Sum of *f_quantity* and *f_totalamount* per *c_name*, *p_name* and *y_yearkey*		*c_name*, *p_name*, *y_yearkey*
	Q13	Sum of *f_quantity* and *f_totalamount* per *c_name*, *p_name* and *y_yearkey*	$y\_yearkey > 2000$ and $c\_acctbal > 5000$	*c_name*, *p_name*, *y_yearkey*
	Q14	Sum of *f_quantity* and *f_totalamount* per *c_name*, *p_name* and *y_yearkey*	$c\_mktsegment =$ "*AUTOMOBILE*" and $y\_yearkey = 2002$	*c_name*, *p_name*, *y_yearkey*
Complex hierarchy	Q15	Avg of *f_quantity* and *f_totalamount* per *t_name*		*t_name*
	Q16	Avg of *f_quantity* and *f_totalamount* per *t_name*	$t\_name =$ "*BRUSHED*"	*t_name*
	Q17	Avg of *f_quantity* and *f_totalamount* per *p_name*	$t\_name =$ "*BRUSHED*"	*p_name*
	Q18	Sum of *f_quantity* and *f_totalamount* per *p_name*	$p\_size > 40$	*p_name*
	Q19	Sum of *f_quantity* and *f_totalamount* per *t_name*	$p\_size > 40$	*t_name*
	Q20	Sum of *f_quantity* and *f_totalamount* per *t_name*	$p\_size > 40$	*t_name*

derive any more complex, composite metrics, such as TPC-H's throughput and power if necessary, while remaining simple.

5 Sample Experiments

To illustrate XWeB's usage, we compare in this section a sample of XML-native DBMSs, namely BaseX[1], eXist[2], Sedna[3], X-Hive[4] and xIndice[5]. We focus on XML-native systems in these experiments because they support the formulation of decision support XQueries that include join operations, which are much more difficult to achieve in XML-compatible relational DBMSs. In these systems, XML documents are indeed customarily stored in table rows, and XQueries are embedded in SQL statements that target one row/document, making joins between XML documents difficult to express and inefficient.

Our experiments atomize the execution protocol from Section 4.3, on one hand to separately outline how its steps perform individually and, on the other hand, to highlight performance differences among the studied systems. Moreover, we vary data warehouse size (expressed in number of facts) in these experiments, to show how the studied systems scale up. Table 2 provides the correspondence between the number of facts, parameters SF and D, and warehouse size in kilobytes. Note that warehouse sizes are small because most of the studied systems *do not* scale up on the hardware configuration we use (a Pentium 2 GHz PC with 1 GB of main memory and an IDE hard drive running under Windows XP). The possibility of missing values and element reordering is also disregarded in these preliminary experiments, i.e., $P_m = P_o = 0$.

5.1 Load Test

Figure 7 represents loading time with respect to data warehouse size. We can cluster the studied systems in three classes. BaseX and Sedna feature the best loading times. BaseX is indeed specially designed for full-text storage and allows compact and high-performance database storage, while Sedna divides well-formed XML documents into parts of any convenient size before loading them into a database using specific statements of the Data Manipulation Language. Both these systems load data about twice faster than X-Hive and xIndice, which implement specific numbering schemes that optimize data access, but require more computation at storage time, especially when XML documents are bulky. Finally, eXist performs about twice worse than X-Hive and xIndice because, in addition to the computation of a numbering scheme, it builds document, element and attribute indexes at load time.

[1] http://www.inf.uni-konstanz.de/dbis/basex/
[2] http://exist.sourceforge.net
[3] http://www.modis.ispras.ru/sedna/
[4] http://www.emc.com/domains/x-hive/
[5] http://xml.apache.org/xindice/

Table 2. Total size of XML documents

SF	D	Number of facts	Warehouse size (KB)
1	$1/14 \times 10^{-7}$	500	1710
1	$1/7 \times 10^{-7}$	1000	1865
1	$2/7 \times 10^{-7}$	2000	2139
1	$3/7 \times 10^{-7}$	3000	2340
1	$4/7 \times 10^{-7}$	4000	2686
1	$5/7 \times 10^{-7}$	5000	2942
1	$6/7 \times 10^{-7}$	6000	3178
1	10^{-7}	7000	3448

5.2 Performance Test

In this set of experiments, we measure query execution time with respect to data warehouse size. Since we rapidly test the limits of the studied systems, we only and separately evaluate the response of reporting, 1-dimension cube, and complex hierarchy-based queries, respectively. In terms of workload parameters, $RE = 1D = CH = TRUE$ and $2D = 3D = FALSE$. Moreover, we stop time measurement when workload execution time exceeds three hours. Finally, since we perform atomic performance tests, they are only cold runs (i.e., $NRUN = 0$).

Figure 8 represents the execution time of reporting queries (RE) with respect to warehouse size. Results clearly show that X-Hive's claimed scalability capability is effective, while the performance of other systems degrades sharply when warehouse size increases. We think this is due to X-Hive's specifically designed XProc query Engine (a pipeline engine), while Sedna and BaseX are specially designed for full-text search and do not implement efficient query engines for structural query processing. Finally, eXist and xIndice are specifically adapted to simple XPath queries processed on a single document and apparently do not suit complex querying needs.

In Figure 9, we plot the execution time of 1D cube queries (1D) with respect to warehouse size. We could only test Sedna and X-Hive here, the other systems being unable to execute this workload in a reasonable time (less than three hours). X-Hive appears the most robust system in this context. This is actually why we do not push toward the 2D and 3D performance tests. Only X-Hive is able to execute these queries. With other systems, execution time already exceeds three hours for one single query. The combination of join and grouping operations (which induce further joins in XQuery) that are typical in decision support queries should thus be the subject of dire optimizations.

Finally, Figure 10 features the execution time of complex hierarchy-based queries (CH) with respect to warehouse size. In this test, we obtained results

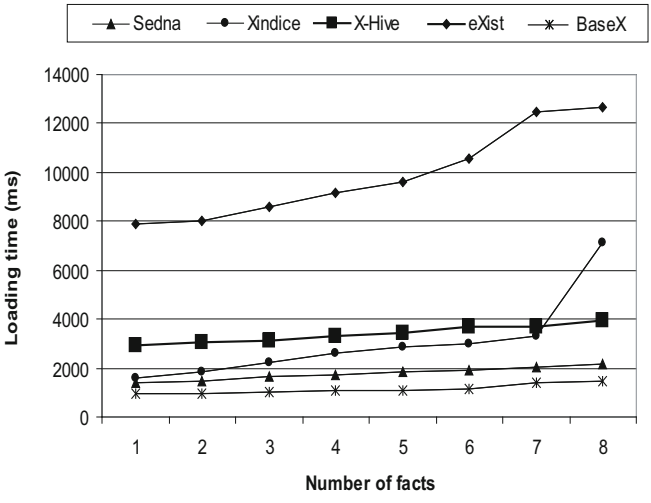

Fig. 7. Load test results

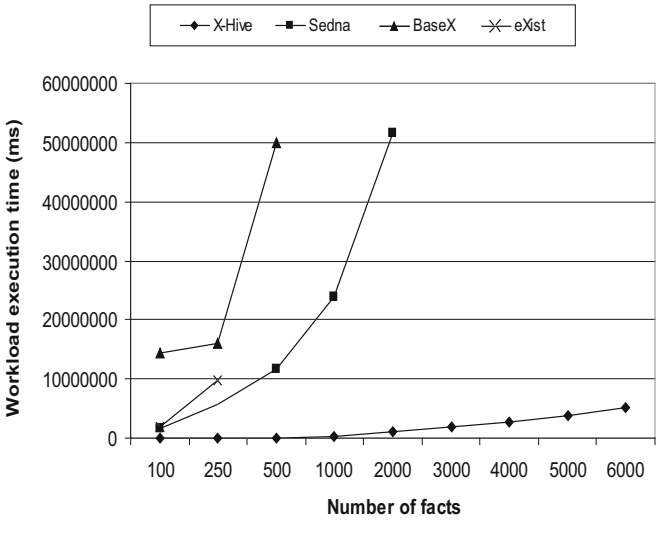

Fig. 8. RE performance test results

only with X-Hive, Sedna and BaseX. Again, X-Hive seems the only XML-native DBMS to be able to scale up with respect to warehouse size when multiple join operations must be performed.

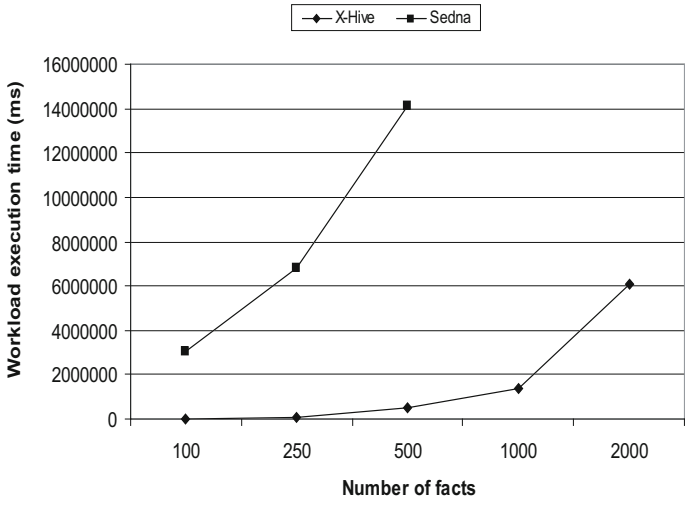

Fig. 9. 1D performance test results

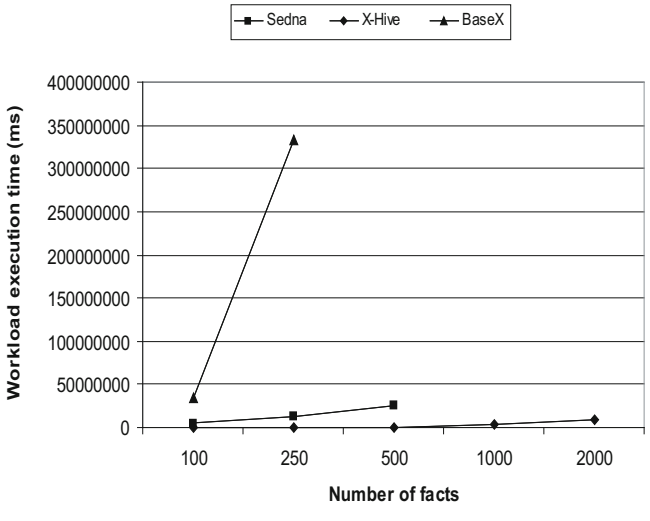

Fig. 10. CH performance test results

6 Conclusion and Perspectives

When designing XWeB, which is to the best of our knowledge the first XML decision support benchmark, we aimed at meeting the four key criteria that make a "good" benchmark according to Jim Gray [8]. *Relevance* means the benchmark must answer various engineering needs. This is why we chose to base

our work on a TPC standard. We also introduced more tunability, both at schema and workload levels, to adapt to the reality of XML DBMSs. *Portability* means the benchmark must be easy to implement on different systems. To this aim, we implemented XWeB with the Java language that allows connecting to most XML DBMSs through APIs (we used the very popular XML:DB[6]). *Scalability* means it must be possible to benchmark small and large databases, and to scale up the benchmark, which is achieved by inheriting from the SF parameter. Further tuning is achieved through the density (D) parameter. Eventually, *simplicity* means that the benchmark must be understandable, otherwise it will not be credible nor used. This is why we elected to base XWeB on TPC-H rather than TPC-DS or DWEB.

In this paper, we also illustrated XWeB's relevance through several experiments aimed at comparing the performance of five native-XML DBMSs. Although basic and more focused on demonstrating XWeB's features than comparing the studied systems in depth, they highlight X-Hive as the most scalable system, while full-text systems such as BaseX seem to feature the best data storage mechanisms. Due to equipment limitations, we remain at small scale factors, but we believe our approach can be easily followed for larger scale factors. We also show the kind of decision support queries that require urgent optimization: namely, cubing queries that perform join and grouping operations on a fact document and dimension documents. In this respect, XWeB had previously been successfully used to experimentally validate indexing and view materialization strategies for XML data warehouses [13].

Eventually, a raw, preliminary version of XWeB (warehouse, workload, Java interface and source code) is freely available online[7] as an Eclipse[8] project. A more streamlined version is in the pipe and will be distributed under Creative Commons licence[9].

After having designed a benchmark modeling business *data* (which XWeB aims to be), it would be very interesting in future research to also take into account the invaluable business information that is stored into unstructured documents. Hence, including features from, e.g., XBench into XWeB would help improve a decision support benchmark's XML specificity.

Since the XQuery Update Facility has been issued as a candidate recommendation by the W3C [6] and is now implemented in many XML DBMSs (e.g., eXist, BaseX, xDB, DB2/PureXML, Oracle Berkeley DB XML...), it will also be important to include update operations in our workload. The objective is not necessarily to feature full ETL testing capability, which would presumably necessitate a dedicated benchmark (Section 4.1), but to improve workload relevance with refreshing operations that are casual in data warehouses, in order to challenge system response and management of redundant performance optimization structures such as indexes and materialized views.

[6] http://xmldb-org.sourceforge.net/xapi/
[7] http://ena-dc.univ-lyon2.fr/download/xweb.zip
[8] http://www.eclipse.org
[9] http://creativecommons.org/licenses/by-nc-sa/2.5/

The core XWeB workload (i.e., read accesses) shall also be given attention. It has indeed been primarily designed to test scaling up. Filter factor analysis of queries [16] and experimental feedback should help tune it and broaden its scope and representativity. Moreover, we mainly focus on cube-like aggregation queries in this version. Working on the output cubes from these queries might also be interesting, i.e., by applying other usual XOLAP operators such as slice & dice or rotate that are easy to achieve in XQuery [9].

Finally, other performance metrics should complement response time. Beyond composite metrics such as TPC benchmarks', we should not only test system response, but also the quality of results. As we underlined in Section 4.3, complex grouping XQueries may return false answers. Hence, query result correctness or overall correctness rate could be qualitative metrics. Since several XQuery extension proposals do already support grouping queries and OLAP operators [2,9,12,26], we definitely should be able to test systems in this regard.

References

1. Afanasiev, L., Manolescu, I., Michiels, P.: MemBeR: A Micro-benchmark Repository for XQuery. In: Bressan, S., Ceri, S., Hunt, E., Ives, Z.G., Bellahsène, Z., Rys, M., Unland, R. (eds.) XSym 2005. LNCS, vol. 3671, pp. 144–161. Springer, Heidelberg (2005)
2. Beyer, K.S., Chamberlin, D.D., Colby, L.S., Özcan, F., Pirahesh, H., Xu, Y.: Extending XQuery for Analytics. In: 2005 ACM SIGMOD International Conference on Management of Data, SIGMOD 2005, Baltimore, USA, pp. 503–514 (2005)
3. Boag, S., Chamberlin, D., Fernández, M., Florescu, D., Robie, J., Siméon, J.: XQuery 1.0: An XML Query Language (2007), http://www.w3.org/TR/xquery/
4. Böhme, T., Rahm, E.: Multi-user Evaluation of XML Data Management Systems with XMach-1. In: Bressan, S., Chaudhri, A.B., Li Lee, M., Yu, J.X., Lacroix, Z. (eds.) CAiSE 2002 and VLDB 2002. LNCS, vol. 2590, pp. 148–159. Springer, Heidelberg (2003)
5. Bressan, S., Lee, M.L., Li, Y.G., Lacroix, Z., Nambiar, U.: The XOO7 Benchmark. In: Bressan, S., Chaudhri, A.B., Li Lee, M., Yu, J.X., Lacroix, Z. (eds.) CAiSE 2002 and VLDB 2002. LNCS, vol. 2590, pp. 146–147. Springer, Heidelberg (2003)
6. Chamberlin, D., Dyck, M., Florescu, D., Melton, J., Robie, J., Siméon, J.: XQuery Update Facility 1.0 (2009), http://www.w3.org/TR/xquery-update-10/
7. Darmont, J., Bentayeb, F., Boussaid, O.: Benchmarking data warehouses. International Journal of Business Intelligence and Data Mining 2(1), 79–104 (2007)
8. Gray, J.: The Benchmark Handbook for Database and Transaction Processing Systems, 2nd edn. Morgan Kaufmann, San Francisco (1993)
9. Hachicha, M., Mahboubi, H., Darmont, J.: Expressing OLAP operators with the TAX XML algebra. In: 3rd International Workshop on Database Technologies for Handling XML Information on the Web, EDBT/DataX 2008, Nantes, France (2008)
10. Hümmer, W., Bauer, A., Harde, G.: XCube: XML for data warehouses. In: 6th International Workshop on Data Warehousing and OLAP, DOLAP 2003, New Orleans, USA, pp. 33–40 (2003)
11. Kimball, R., Ross, M.: The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling, 2nd edn. Wiley, Hoboken (2002)

12. Kit, C., Amagasa, T., Kitagawa, H.: Algorithms for structure-based grouping in XML-OLAP. International Journal of Web Information Systems 5(2), 122–150 (2009)
13. Mahboubi, H., Darmont, J.: Benchmarking XML data warehouses. In: 1st Workshop on Decisional Systems, MCSEAI/ASD 2006, Agadir, Morocco (2006)
14. Mahboubi, H., Ralaivao, J.C., Loudcher, S., Boussaid, O., Bentayeb, F., Darmont, J.: X-WACoDa: An XML-based approach for Warehousing and Analyzing Complex Data. In: Advances in Data Warehousing and Mining, pp. 38–54. IGI (2009)
15. OLAP Council: APB-1 OLAP Benchmark Release II (1998), http://www.olapcouncil.org
16. O'Neil, P., O'Neil, E., Chen, X., Revilak, S.: The Star Schema Benchmark and Augmented Fact Table Indexing. In: Nambiar, R., Poess, M. (eds.) TPCTC 2009. LNCS, vol. 5895, pp. 237–252. Springer, Heidelberg (2009)
17. Pokorný, J.: XML Data Warehouse: Modelling and Querying. In: 5th International Baltic Conference (BalticDB&IS 2002), Tallin, Estonia, pp. 267–280 (2002)
18. Rizzi, S.: Conceptual Modeling Solutions for the Data Warehouse. In: Data Warehouses and OLAP: Concepts, Architectures and Solutions, pp. 1–26. IRM Press, Hershey (2007)
19. Runapongsa, K., Patel, J.M., Jagadish, H.V., Chen, Y., Al-Khalifa, S.: The Michigan benchmark: towards XML query performance diagnostics. Information Systems 31(2), 73–97 (2006)
20. Schmidt, A., Waas, F., Kersten, M.L., Carey, M.J., Manolescu, I., Busse, R.: Assessing XML Data Management with XMark. In: Bressan, S., Chaudhri, A.B., Li Lee, M., Yu, J.X., Lacroix, Z. (eds.) CAiSE 2002 and VLDB 2002. LNCS, vol. 2590, pp. 144–145. Springer, Heidelberg (2003)
21. Simitsis, A., Vassiliadis, P., Dayal, U., Karagiannis, A., Tziovara, V.: Benchmarking ETL Workflows. In: Nambiar, R., Poess, M. (eds.) TPCTC 2009. LNCS, vol. 5895, pp. 199–220. Springer, Heidelberg (2009)
22. Thomsen, E.: Comparing different approaches to OLAP calculations as revealed in benchmarks. Intelligence Enterprise's Database Programming & Design (1998), http://www.dbpd.com/vault/9805desc.htm
23. Torlone, R.: Conceptual Multidimensional Models. In: Multidimensional Databases: Problems and Solutions, pp. 69–90. IDEA, USA (2003)
24. Transaction Processing Performance Council: TPC Benchmark DS (Decision Support) Draft Specification revision 32 (2005), http://www.tpc.org/tpcds/
25. Transaction Processing Performance Council: TPC Benchmark H Standard Specification Revision 2.8.0 (2008), http://www.tpc.org/tpch/
26. Wiwatwattana, N., Jagadish, H.V., Lakshmanan, L.V.S., Srivastava, D.: X^3: A Cube Operator for XML OLAP. In: 23rd International Conference on Data Engineering, ICDE 2007, Istanbul, Turkey, pp. 916–925 (2007)
27. Wyatt, L., Caufield, B., Pol, D.: Principles for an ETL Benchmark. In: Nambiar, R., Poess, M. (eds.) TPCTC 2009. LNCS, vol. 5895, pp. 183–198. Springer, Heidelberg (2009)
28. Yao, B.B., Özsu, M.T., Khandelwal, N.: XBench Benchmark and Performance Testing of XML DBMSs. In: 20th International Conference on Data Engineering, ICDE 2004, Boston, USA, pp. 621–633 (2004)
29. Zhang, J., Wang, W., Liu, H., Zhang, S.: X-warehouse: building query pattern-driven data. In: 14th International Conference on World Wide Web, WWW 2005, Chiba, Japan, pp. 896–897 (2005)

Benchmarking Using Basic DBMS Operations

Alain Crolotte and Ahmad Ghazal

Teradata Corporation, 100 N Sepulveda Blvd.
El Segundo, Ca. 90045
{alain.crolotte,ahmad,ghazal}@teradata.com

Abstract. The TPC-H benchmark proved to be successful in the decision support area. Many commercial database vendors and their related hardware vendors used these benchmarks to show the superiority and competitive edge of their products. However, over time, the TPC-H became less representative of industry trends as vendors keep tuning their database to this benchmark-specific workload. In this paper, we present XMarq, a simple benchmark framework that can be used to compare various software/hardware combinations. Our benchmark model is currently composed of 25 queries that measure the performance of basic operations such as scans, aggregations, joins and index access. This benchmark model is based on the TPC-H data model due to its maturity and well-understood data generation capability. We also propose metrics to evaluate single-system performance and compare two systems. Finally we illustrate the effectiveness of this model by showing experimental results comparing two systems under different conditions.

1 Introduction

In the DSS area the TPC-D then TPC-H benchmarks [1] were quite successful as demonstrated by the large number of companies that ran them. The 3rd normal form schema on which it is based is easy to understand and rich enough to perform interesting experiments. The main advantage provided by this benchmark has been the data generation utility (known as dbgen) that guaranteed repeatability on all platforms. The benchmark is now showing its age and its usefulness to drive the development of new database features is severely diminished. This is particular true in the area of query optimization as the query set is fixed and at this point in time too well understood. It still remains today a de facto standard and its good features are still very useful. In particular, it can be used as a basis for developing interesting workloads that are related but different from TPC benchmarks are used extensively in research by universities [2] or corporations [3, 9,10].

There are several issues associated with the TPC-H benchmark aside from its age. The main problem is the set of rules and restrictions that govern its execution. New database features often need to be adapted to be transparent or hidden to allow publication. As a result of this lack of transparency it is often difficult to assess what techniques are actually used in a published result and how effective these techniques are. A side effect of this state of affairs is an inherent unfairness in comparing results. While shortly after the inception of the TPC-D benchmark it actually made sense for

R. Nambiar and M. Poess (Eds.): TPCTC 2010, LNCS 6417, pp. 204–215, 2011.

vendors to partly assess new hardware or a new software releases with TPC-D metrics, it is no longer the case. It was still possible after the inception of the TPC-H benchmark as the few changes over TPC-D kept the interest alive for a while. It is still somewhat possible to get a glimpse at a new architecture by looking at the individual times in the power test for queries 1 and 6 since usually, they are respectively CPU and I/O bound.

In this paper, we propose a very simple benchmark approach based on the TPC-H workload allowing a global and fair evaluation of new features. This approach could be of interest especially for hardware vendors as well. Hardware vendors have many tools at their disposal to determine the raw power of their equipment but these approaches usually do not take into account the database layer. What we propose here instead is an approach that takes the database layer into account.

Many of the database optimization techniques designed for performance actually avoid utilizing the hardware. A good example is materialized structures such as materialized views, join indexes or summary tables since they practically bypass the I/O subsystem. Our benchmark is designed in such a way to exercise fully the hardware through basic database functions such as scans, joins and aggregations. Although the benchmark presented can be used "as is" the approach can also be adapted to define more involved queries that would lend themselves to comparisons between an optimized and a non-optimized workload.

There were several attempts to define new benchmarks around TPC-D and TPC-H Some of these attempts modified the data model or added a missing element or added new queries. In general it would be easy to introduce skew with the customer or supplier distributions as was done in [6] a private benchmark built on top of TPC-D that introduced skew. Another example is [7] in which the authors present a star schema around the TPC-H benchmark. We follow a similar approach without modifying the TPC-H schema nor the data generation utility.

In section 2 we present XMarq, our benchmark proposal. The discussion is broken into different database classes such as scans and aggregations. This includes a description of the queries to be run in each class and the rules under which these queries should be run. In section 3 we provide benchmark results and propose a methodology to evaluate those results absolutely and comparatively. In section 4 we analyze the XMarq benchmark in light of ideal conditions proposed at the 2009 TPC Conference [8]. The actual SQL associated with the queries is provided in the Appendix.

2 Benchmark Specifications

The TPC-H data model is a retail model fully described in [1]. TPC-H consists of 8 tables with history over 8 years. We are presenting queries using mainly ORDERS (primary key (PK) =o_orderkey) containing the orders, LINEITEM (PK=l_orderkey, l_linenumber) containing the line items associated with these ORDERS and Customer (PK=c_custkey) contains customers placing orders. Customers have a nation they belong to (FK=c_nationkey). There is a PART table (PK=p_partkey) and LINEITEM has a FK=l_partkey.

Even though the XMarq benchmark can be defined for any scale factor supported by dbgen, not just the official TPC-H scale, we will define it here with scale factor

1000 for the sake of simplicity. XMarq is designed to be run right after the database has been materialized with dbgen. Because the benchmark must represent an ad hoc environment, only primary and foreign key columns may be indexed, and no optimizer hints or extra pre-calculated structures may be used. Also pre-staging of the data including indexes is disallowed and so is vertical partitioning.. Only raw system performance through simple database queries is to be measured not the sophistication of the optimizer. Similarly, no techniques to limit the number of rows returned are allowed.

The benchmark is organized into five query groups: scans, aggregations, joins, CPU-intensive and indexed. The main idea is to utilize known properties of the TPC-H data to define basic queries. For a full table scan for instance we can use a constraint that is known to be impossible to satisfy. Examples of this can be l_linenumber < 0 since we know that that field is always between 1 and 7. In the sequel we go trough the proposed queries. Queries are named with a 2-letter code in which the first letter identifies the query group (e.g. S to scans, A for aggregation).

2.1 Scan Queries

ST – LARGE TABLE SCAN
This query consists of scanning the LINEITEM table while returning no rows This is accomplished by selecting all the columns based on a constraint that is never satisfied namely l_linenumber < 0. For this query we assume that there are no statistics or index on l_linenumber.

SI – MATCHING INSERT/SELECT
This query populates PARTX a copy of the PART table distributed by p_partkey with an insert/select. Matching merely means that the original table and its copy use the same DDL including the distribution field, in this case p_partkey.

SN – NON MATCHING INSERT/SELECT
This query populates PARTX with DDL similar to PART but distributed differently, in this case, by the non-unique combination of p_size, p_brand and p_container instead of p_partkey. This will likely cause more data movement than the matching insert/select in most products.

SU – SCAN UPDATE 4% OF THE ROWS
This query utilizes the fact that column p_brand in PARTX has only 25 distinct vales so that, due to the uniform distribution of data produced by dbgen, any specific value of p_brand will be represented in about 4% of the rows.

SP – SCAN UPDATE 20% OF THE ROWS
This query utilizes the fact that p_mfgr has only 5 distinct values. As a result a particular value of p_mfgr will involve about 20% of the rows. At scale factor 1000 exactly 40,006,935 rows out of 200 millions are updated.

2.2 Aggregation Queries

AR – ROW COUNT
This query does a simple count of the rows in the largest table in the database, LINEITEM. The number of rows returned is well-documented in the TPC-H spec.

AD – DISTINCT COUNT
This query counts the distinct values of the column l_quantity in the LINEITEM table. A single row with a value of 50 should be returned at all scale factors and l_quantity must not be indexed for this query.

AS – 15-GROUP AGGREGATE
This query utilizes ORDERS and the fact that the combination o_ordertatus, o_orderpriority has only 15 distinct combinations at all scale factors.

AM – THOUSAND GROUP AGGREGATE
This query uses LINEITEM and l_receipdate that has only a limited number of values (2555 at scale factor 1000). While l_shipdate is more predictable (exactly 2406 distinct values at all scale factors) this field plays too central of a role in the TPC-H queries. No index should be placed on l_receiptdate for this query.

AL – HUNDRED THOUSAND GROUP AGGREGATE
This query is meant to build over 100000 aggregate groups. By using the first 15 characters of o_comment one can build exactly 111517 groups at scale factor 1000. To further limit the number of rows actually retrieved we added a limit on o_totalprice.

2.3 Join Queries

JI – IN-PLACE JOIN
In this query we join ORDERS and LINEITEM on the key common to both tables without constraints while performing a calculation ensuring that the join is performed but only one row is returned.

JF – PK/FK JOIN
This query joins PART and LINEITEM on partkey which is the PK for PART and the FK for LINEITEM while performing an operation involving columns in both tables. No index on l_partkey is allowed for this query.

JA – AD-HOC JOIN
This query joins PART and LINEITEM on unrelated columns (p_partkey and l_suppkey) while performing a sum so that only one row is returned. Because of the fact that the join columns are sequential integers there will be some matching.

JL – LARGE/SMALL JOIN
This query joins CUSTOMER and NATION on nationkey while performing a group by operation. This is also an FK/PK join but the salient feature here is the size discrepancy between the tables since NATION has only 25 rows.

JX – EXCLUSION JOIN
This query calculates the total account balance for the one-third of customers without orders. For those customers the CUSTOMER rows do not have a customer key entry in the ORDER table. The inner query selects customers that do have orders so that by leaving these out we get the customers that do not have orders.

2.4 CPU Intensive Queries

CR – ROLLUP REPORT
This query covers the group by operator. To accomplish that, the query is applied on a single table with a simple predicate. The query involves 7 columns and 12 aggregations to make it CPU intensive.

CF – FLOATS & DATES
This query maximizes CPU activity while minimizing disk or interconnect overhead. Floating point conversion is done multiple times for each of the rows, as well as repetitive complex date conversions. Because of its emphasis on repetitive CPU-intensive operations, it will highlight products that offer machine readable evaluation code, as opposed to the more common interpretive code. It will also tend to favor products that are capable of caching and reusing repetitive instructions, and are coded to perform such activities efficiently. To minimize I/O, a predicate is used to filter out all but one row.

2.5 Index Queries

IP – PRIMARY RANGE SEARCH
This query measures the ability of database system to select from a table based on a range of values applied to the table's primary (or clustering) index. In this case the range constraint is on the same column as the partitioning key (clustering, primary index). It is designed to highlight capabilities such as value-ordered or value-sensitive indexes.

A second constraint has been added to the WHERE clause, based on P_RETAILPRICE. Its purpose is to prevent large numbers of rows from being returned. An index on this constraint column is disallowed, so as to make sure that the primary index is the only index available for data access.

This primary key range search query is different from the next query, I2, which uses secondary index access, because many database systems organize these different types of indexes differently, with differing performance characteristics. Often a primary index will determine the physical location or the physical sequencing of a row within a table. Some database systems, however, do not differentiate between partitioning (or primary) indexes and secondary indexes. This query returns 44 rows at Scale Factor 1000.

IR – SECONDARY RANGE SEARCH
In this query, the range constraint column is not the same column as the partitioning key, as was the case with query IP above. Secondary index access usually requires traversing a secondary structure which points to the location of the physical rows

required. This physical structure may or may not be sequenced, or may or may not be efficient to navigate. This query attempts to measure those differences. This query is expected to favor products that have value-ordered secondary indexes, or cost-based optimizers intelligent enough to consider scanning an index structure. An index must be placed on l_shipdate for this query.

IL – LIKE OPERATOR

This query uses the text string search capabilities of the LIKE clause, searching for all rows that have a value in their O_CLERK column that begin with that string. It then returns each distinct value for the O_CLERK column satisfies the LIKE clause condition.

The number of distinct clerks in the Order table is determined at data generation time by multiplying the scale factor by 1000. For Scale Factor 10 there will be 10,000 Clerks randomly assigned to ORDERS. As there are 9 significant digits making up O_CLERK, and 7 of them are to the left of the percent sign (with 2 remaining, we know there are a possible 100 Clerks that meet the criteria of this query.

The query returns 100 rows at all scale factors. An index must be placed on o_clerk for this query.

IB – BETWEEN OPERATOR

This query uses a third type of constraint with an index: the between clause. This query requests from the LINEITEM table only rows that have a ship date of a single month in 1992. After accessing those particular LINEITEM rows, it returns only the distinct quantities associated with the rows l_quantity as a column is a random value from 1 to 50. As there are approximately 7 years of data at all volume points, one month constitutes 1/84, or 1.2% of the LINEITEM rows which are processed by this query. Similarly to query IR an index must be placed on l_shipdate for this query.

II – MULTIPLE INDEX ACCESS

While previous index access queries tested either primary or secondary index strategies, this query combines them both.

This query utilizes the OR operator between two constraints: one constraint is on a non-unique index on the Clerk column (as we mentioned above, there are 10,000 Clerks at Scale Factor 10, about 1500 orders have the same Clerk) and the second is on a unique index o_orderkey. There are 1462 actual Clerks with this Clerk_ID in the ORDERS table.

Since this is OR condition, a row will qualify for inclusion in the answer set if either side of the OR is satisfied, so some database optimizer decisions will need to accommodate this possible complexity. This query tests the flexibility and diversity of the optimizer in a controlled environment with mandatory indexes.

Some database systems might be able to satisfy this query by only traversing the index structures, and not actually doing physical I/O on the base table itself. This is a possibility, in part, because of the query's select list. The query only returns a count of how many rows pass either of the selection criteria, and does not request data from the base table itself. A single row with a count of 1462 is returned. An multiple column index on o_clerk and o_orderkey must be exist for this query.

IC – COUNT BY INDEX
This query lists each distinct Discount in the largest table in the database, and counts how many rows are associated with each of the Discounts it finds. This is a common activity useful to validate the database information, or to find out how your customer base is skewed geographically.

This query is both a simple aggregation query and a query that demonstrates flexibility in the use of the secondary index structure. Assuming an index is placed on the l_discount column, it may be possible to accommodate this request without having to read the base table. This greatly reduces I/O overhead in this case, because this query can only be satisfied by looking at information stored in the index structure and simply counting rows from there. There are 11 different discount values in l_discount. Therefore this query will return 11 rows for all Scale Factors.

IM – MULTI-COLUMN INDEX -- LEADING VALUE ONLY
It is not uncommon that in index created for a broader purpose may be useful for queries other than for which it was intended. In this query we have a 2-column index on p_type and p_size but the condition is on p_type only.

This query tests the ability of the index to respond to broader needs than anticipated. There are 150 different p_type values in the PART table. This query determines what the average retailprice is for all the Parts of that type. A single row is returned by this query

IT – MULTI-COLUMN INDEX -- TRAILING VALUE ONLY
A somewhat related need is measured by this query, which provides a value for the second, or trailing value in a 2-column index. There are 50 distinct values in p_size, of which only one is selected here. This query, similar to the one above, averages the retailprice in all qualifying parts that have the size of 21. Instead of accessing the leading field of the 2-column index p_size, p_type this query accesse the trailing field of the index. Only a single row is returned.

3 Experimental Results

In order to illustrate the use of the benchmark in a real-life case we ran the benchmark with the same version of Teradata on two different platforms. The columns marked system A and system B portray actual performance numbers. In those columns the summary per category is an arithmetic mean. For instance, for scans, what we have in row SCANS is the arithmetic mean of all scan type performance for system A and system B respectively. The numbers in column A/B are ratios and the method to aggregate those numbers is the geometric mean (see [4] and [5] for an explanation of the definition and proper use of arithmetic and geometric means). For instance, for scans, the score of 4.8 is the geometric mean of the five numbers above. The overall scores follow the same logic. The overall score for system A and system B are the arithmetic means of their individual raw performance at the individual query level while the overall ratio score is the geometric mean of all ratios at query level.

QUERIES/CATEGORIES	System A	System B	A/B
ST -- LARGE TABLE SCAN	95.9	18.6	5.1
SI -- MATCHING INSERT/SELECT	48.0	8.7	5.5
SN -- NON-MATCHING INSERT/SELECT	174.6	25.0	7.0
SU -- SCAN/UPATE 4% OF THE ROWS	191.5	58.4	3.3
SP -- SCAN/UPATE 20% OF THE ROWS	906.3	244.1	3.7
SCANS	**283.2**	**71.0**	**4.8**
AR -- ROW COUNT	0.8	0.3	3.0
AD -- COUNT DISTINCT VALUES	112.3	29.8	3.8
AS -- 15-GROUP AGGREGATE	51.5	14.4	3.6
AM -- THOUSAND-GROUP AGGREGATE	30.2	9.7	3.1
AL -- HUNDRED-THOUSAND GROUP AGGREGATION	61.1	16.6	3.7
AGGREGATIONS	**51.1**	**14.2**	**3.4**
JI -- IN PLACE JOIN	372.8	123.3	3.0
JF -- FOREIGN KEY TO PRIMARY KEY JOIN	2441.5	340.5	7.2
JA -- AD HOC JOIN	891.9	191.9	4.6
JL -- LARGE/SMALL JOIN	0.1	0.0	11.0
JX -- EXCLUSION JOIN	21.3	19.6	1.1
JOINS	**745.5**	**135.1**	**3.2**
CR -- ROLLUP REPORT	312.3	75.2	4.2
CF -- FLOATS & DATES	96.4	16.5	5.9
CPU INTENSIVE	**204.3**	**45.8**	**4.9**
IP -- PRIMARY RANGE SEARCH	3.1	1.0	3.3
IR -- SECONDARY RANGE SEARCH	10.7	0.3	42.7
IL -- LIKE OPERATOR	29.0	7.7	3.8
IB -- BETWEEN OPERATOR	14.1	0.6	22.1
II -- MULTIPLE INDEX ACCESS	29.0	7.5	3.9
IC -- COUNT BY INDEX	6.5	1.8	3.6
IM -- MULTI-COLUMN INDEX -- LEADING VALUE ONLY	1.9	0.3	5.6
IT -- MULTI-COLUMN INDEX -- TRAILING VALUE ONLY	1.7	0.2	7.3
INDEXED OPERATIONS	**12.0**	**2.4**	**7.1**
overall scores	***236.2***	***48.5***	***4.9***

4 Conclusions

In [8] Huppler proposed five criteria for a good benchmark. The most important one is "relevance" and the other four are "repeatability", "fairness", "verifiability" and "being economical". Huppler also made the point that there should be a balance or

tradeoff between "relevance" and the other four criteria. We believe that XMarq meets all five criteria. In terms of relevance XMarq scores very high since it focuses on the most common paths in the database code. It also can be run on any hardware supporting the database under test. XMarq is repeatable since it is based on the well-established TPC-H schema, the dbgen code that produces repeatable data and very simple SQL, actually a subset of the TPC-H SQL. These qualities make XMarq easily repeatable and portable. XMarq is also fair since it focuses on basic SQL operations and does not favor exotic optimizer features that are specific to certain database products. Verifying XMarq results is also straightforward. The metric proposed is actually better than the TPC-H power metric since it uses the arithmetic mean while the geometric mean is used appropriately for ratios i.e. normalized data. Other measurements can be performed e.g. CPU, I/O or network consumption. XMarq is meant to measure the basic performance of existing hardware and software and therefore does not require any additional development. Finally, the overall cost is much lower than TPC-H since the process of running and auditing the benchmark is straightforward. It would be very interesting for hardware vendors to correlate TPC-H and XMarq results for a given database.

References

1. TPC Benchmark D and H (Decision Support), Transaction Processing Council, http://www.tpc.org
2. Gao, K., Pandis, I.: Implementation of TPC-H and TPC-C toolkits, CS and ECE Departments, Carnegie Mellon University, http://www.cs.cmu.edu/~ipandis/courses/15823/project_final_paper.pdf
3. Ghazal, A., Seid, D., Ramesh, B., Crolotte, A., Koppuravuri, M., Vinod, G.: Dynamic plan generation for parameterized queries. In: Proceedings of the 35th SIGMOD International Conference on Management of Data, SIGMOD, pp. 909–916 (2009)
4. Crolotte, A.: Issues in Metric Selection and the TPC-D Single Stream Power, http://www.tpc.org
5. Crolotte, A.: Issues in Benchmark Metric Selection. In: Nambiar, R.O., Poess, M. (eds.) TPCTC 2009. LNCS, vol. 5895, pp. 146–152. Springer, Heidelberg (2009)
6. The Data Warehouse Challenge, Specification document revision 1.4.c, Data Challenge Inc. (April 9, 1998)
7. O'Neil, P., O'Neil, E., Chen, X., Revilak, S.: The Star Schema Benchmark and Augmented Fact Table Indexing. In: Nambiar, R.O., Poess, M. (eds.) TPCTC 2009. LNCS, vol. 5895, pp. 237–252. Springer, Heidelberg (2009)
8. Huppler, K.: The Art of Building a Good Benchmark. In: Nambiar, R.O., Poess, M. (eds.) TPCTC 2009. LNCS, vol. 5895, pp. 18–30. Springer, Heidelberg (2009)
9. Ballinger, C.: Relevance of the TPC-D Benchmark Queries: The Questions You Ask Every Day, NCR Parallel Systems, http://www.tpc.org/information/other/articles/TPCDart_0197.asp
10. Poess, M., Floyd, C.: New TPC Benchmarks for Decision Support and Web Commerce. In: SIGMOD, pp. 64–71 (2000)

Appendix

ST – LARGE TABLE SCAN
SELECT * FROM LINEITEM WHERE L_LINENUMBER < 0;

SI – MATCHING INSERT/SELECT
INSERT INTO PARTX SELECT * FROM PART;

SN – NON MATCHING INSERT/SELECT
INSERT INTO PARTX SELECT * FROM PART;

SP – SCAN UPDATE 4% OF THE ROWS
UPDATE PARTX SET P_RETAILPRICE = (P_RETAILPRICE + 1) WHERE P_BRAND = 'Brand#23';

SP – SCAN UPDATE 20% OF THE ROWS
UPDATE PARTX SET P_RETAILPRICE = (P_RETAILPRICE + 1) WHERE P_MFGR = 'Manufacturer#5';

AR – ROW COUNT
SELECT COUNT(*) FROM LINEITEM;

AD – DISTINCT COUNT
SELECT COUNT (DISTINCT L_QUANTITY) FROM LINEITEM;

AS – 15-GROUP AGGREGATE
SELECT O_ORDERSTATUS, O_ORDERPRIORITY,
AVERAGE (O_TOTALPRICE FROM ORDERS GROUP BY 1, 2;
AM – THOUSAND-GROUP AGGREGATE
SELECT L_RECEIPTDATE, COUNT (*) FROM LINEITEM
GROUP BY 1 ORDER BY 1;

AL – HUNDRED THOUSAND GROUP AGGREGATE
SELECT SUBSTRING (O_COMMENT,1,15), ,COUNT(*) FROM ORDERS
GROUP BY 1;

JI – IN-PLACE JOIN
SELECT AVERAGE (L_QUANTITY) FROM LINEITEM, ORDERS
WHERE L_ORDERKEY=O_ORDERKEY;

JF – PK/FK JOIN
SELECT AVERAGE (P_RETAILPRICE*L_QUANTITY)
FROM PART, LINEITEM WHERE P_PARTKEY=L_PARTKEY;

JA – AD-HOC JOIN
SELECT SUM(L_QUANTITY) FROM PART, LINEITEM
WHERE P_PARTKEY=L_SUPPKEY;

JL – LARGE/SMALL JOIN

```
SELECT N_NAME, AVERAGE(C_ACCTBAL) FROM CUSTOMER, NATION
WHERE C_NATIONKEY=N_NATIONKEY GROUP BY N_NAME;
```

CR – ROLLUP REPORT

```
SELECT L_RETURNFLAG,
L_LINESTATUS,
L_SHIPMODE,
SUBSTRING (L_SHIPINSTRUCT, 1, 1),
SUBSTRING (L_LINESTATUS, 1, 1),
((L_QUANTITY - L_LINENUMBER) + (L_LINENUMBER - L_QUANTITY)),
(L_EXTENDEDPRICE - L_EXTENDEDPRICE),
SUM ((1 + L_TAX) * L_EXTENDEDPRICE),
SUM ((1 - L_DISCOUNT) * L_EXTENDEDPRICE),
SUM (L_DISCOUNT / 3),
SUM (L_EXTENDEDPRICE * (1 - L_DISCOUNT) * (1 + L_TAX)),
SUM (L_EXTENDEDPRICE - ((1 - L_DISCOUNT) * L_EXTENDEDPRICE)),
SUM (DATE - L_SHIPDATE + 5),
SUM (L_SHIPDATE - L_COMMITDATE),
SUM (L_RECEIPTDATE - L_SHIPDATE),
SUM (L_LINENUMBER + 15 - 14),
SUM (L_EXTENDEDPRICE / (10 - L_TAX)),
SUM ((L_QUANTITY * 2) / (L_LINENUMBER * 3)),
COUNT (*)
FROM LINEITEM
WHERE L_LINENUMBER GT 2

GROUP BY
 L_RETURNFLAG,
 L_LINESTATUS,
 L_SHIPMODE,
 SUBSTRING (L_SHIPINSTRUCT,1,1),
 SUBSTRING (L_LINESTATUS,1,1),
  ((L_QUANTITY - L_LINENUMBER) + (L_LINENUMBER - L_QUANTITY)),
  (L_EXTENDEDPRICE - L_EXTENDEDPRICE);
```

CF - FLOATS & DATES

```
SELECT COUN T(*) FROM LINEITEM
WHERE (L_QUANTITY = 1.1E4
OR L_QUANTITY = 2.1E4
OR L_QUANTITY = 3.1E4
OR L_QUANTITY = 4.1E4
OR L_QUANTITY = 5.1E4
OR L_QUANTITY = 6.1E4
OR L_QUANTITY = 7.1E4
OR L_QUANTITY = 8.1E4
OR L_QUANTITY = 9.1E4
```

```
OR L_QUANTITY = 50)
AND (DATE - L_SHIPDATE) GT 0
AND (L_COMMITDATE + 5) LT (L_RECEIPTDATE + 5)
AND (L_SHIPDATE + 20) LT (L_COMMITDATE + 20);
```

IP – PRIMARY RANGE SEARCH

```
SELECT P_NAME, P_RETAILPRICE FROM PART
WHERE P_PARTKEY LT 50000
AND    P_RETAILPRICE LT 909.00;
```

IR – SECONDARY RANGE SEARCH

```
SELECT L_ORDERKEY, L_LINENUMBER
FROM LINEITEM
WHERE L_SHIPDATE LT 981200;
```

IL – LIKE OPERATOR

```
SELECT DISTINCT O_CLERK
FROM ORDERS
WHERE O_CLERK LIKE 'Clerk#0000067%';
```

IB – BETWEEN OPERATOR

```
SELECT DISTINCT L_QUANTITY
FROM LINEITEM
WHERE L_SHIPDATE BETWEEN 930301 AND 930331;
```

II – MULTIPLE INDEX ACCESS

```
SELECT COUNT (*) FROM ORDERS
WHERE O_CLERK = 'CLERK#000006700' OR O_ORDERKEY = 50500;
```

IC – COUNT BY INDEX

```
SELECT L_DISCOUNT, COUNT (*) FROM LINEITEM
GROUP BY L_DISCOUNT;
```

IM – MULTI-COLUMN INDEX -- LEADING VALUE ONLY

```
SELECT AVERAGE (P_RETAILPRICE) FROM PART
WHERE P_TYPE = 'SMALL PLATED BRASS';
```

IT – MULTI-COLUMN INDEX -- TRAILING VALUE ONLY

```
SELECT AVERAGE (P_RETAILPRICE) FROM PART
WHERE P_SIZE = 21;
```

Assessing and Optimizing Microarchitectural Performance of Event Processing Systems

Marcelo R.N. Mendes, Pedro Bizarro, and Paulo Marques

CISUC, University of Coimbra,
Dep. Eng. Informática – Pólo II, 3030-290, Coimbra, Portugal
{mnunes,bizarro,pmarques}@dei.uc.pt

Abstract. Event Processing (EP) systems are being progressively used in business critical applications in domains such as algorithmic trading, supply chain management, production monitoring, or fraud detection. To deal with high throughput and low response time requirements, these EP systems mainly use the CPU-RAM sub-system for data processing. However, as we show here, collected statistics on CPU usage or on CPU-RAM communication reveal that available systems are poorly optimized and grossly waste resources. In this paper we quantify some of these inefficiencies and propose cache-aware algorithms and changes on internal data structures to overcome them. We test the before and after system both at the microarchitecture and application level and show that: i) the changes improve microarchitecture metrics such as clocks-per-instruction, cache misses or TLB misses; ii) and that some of these improvements result in very high application level improvements such as a 44% improvement on stream-to-table joins with 6-fold reduction on memory consumption, and order-of-magnitude increase on throughput for moving aggregation operations.

Keywords: Benchmarking, Complex Event Processing, Performance, Tuning.

1 Introduction

Previous work by Ailamaki [2], Ravishankar [11], and Abadi [1] showed that microarchitecture inspired improvements such as cache-aware algorithms and changes of internal data representations can lead to high improvements on the performance of database or data warehouse systems. Encouraged by this work, we took a similar position and set-out to discover the microarchitecture performance of Event Processing (EP) systems.

Using Esper [6], a widely used open-source EP system, we measured the systems' performance executing common operations such as moving aggregations and stream-to-table joins. We monitored the system at the microarchitecture level using the *Intel VTune*® profiler [8] and also collected application-level metrics such as memory consumption and peak sustained throughput.

To isolate from secondary effects, we then replicated the main algorithms and data structures on our own event processing prototype and progressively improved them

R. Nambiar and M. Poess (Eds.): TPCTC 2010, LNCS 6417, pp. 216–231, 2011.

with microarchitecture-aware optimizations. These optimizations were then validated first by running the tuned prototype in a multi-query scenario, and then porting the modifications back into Esper.

Summary of Contributions

In this paper we make the following contributions:

- We analyzed how current event processing systems perform at both application and hardware levels. By collecting and correlating metrics such as throughput, CPI, and cache misses during execution of continuous queries, we show that microarchitectural aspects significantly influence the final performance of common event processing tasks, being in some cases the sole cause for performance degradation when the input is scaled up (Section 3.3).
- We demonstrated how alternate data structures can drastically improve performance and resource consumption in EP systems (Section 3.4).
- We implemented, tested and evaluated an adapted version of the Grace Hash algorithm [9] for joining event streams with memory-resident tables. Results revealed that by reducing the impact of microarchitectural aspects on query execution performance was improved in up to 44 percent (Section 3.5).
- We implemented, tested and evaluated a microarchitecture-aware algorithm for computing moving aggregations over sliding windows, which provided performance gains ranging from 28 to 35 percent (Section 3.5).

2 Background

From the microarchitectural point of view, the amount of time a given computational task *T* takes to complete depends primarily on two factors: the task size (i.e., Instruction Count or IC) and the average duration of instructions (frequently expressed as Cycles per Instruction or *CPI*). Algebraically, in cycles [7]:

$$CPU\ execution\ time = IC \times CPI$$

Better performance can be achieved by reducing either factor or both. Traditionally, software developers have focused on reducing IC by improving time complexity of algorithms, but an increased interest in making a more efficient use of hardware resources has been observed over the last years [2, 3, 12, 13].

To understand how these optimizations targeted at the hardware level work, it is necessary to know the internals of CPU operation. Every single instruction is executed inside the processor as series of sequential steps across its several functional units. During this sequence of steps, generally referred as *pipeline*, CPU instructions are fetched from memory, decoded, executed and finally have their results stored back into registers or memory. To increase throughput, instructions in different stages/functional units are processed in parallel (*Instruction-Level Parallelism*). In ideal conditions, the processor pipeline remains full most of the time, retiring one or more

instructions per cycle (implying CPI ≤ 1). Many factors, however, can cause instructions to *stall*, thus increasing average CPI. The gap between processor and memory speeds is one of them: an instruction that access data resident in main memory may require tens to hundreds of CPU cycles to complete. Another typical cause of stalls is *data dependency*: when an instruction *j* depends on a value produced by an instruction *i* that was fetched closely before it, *j* cannot execute until *i* completes (in fact, if the processor issues instructions in program order, when instruction *j* stalls, no later instructions can proceed, thus aggravating the performance impact of the data dependency). Finally, *control dependencies*, which happen when the instruction flow cannot be determined until a given instruction *i* (e.g., a conditional branch) completes, can also adversely affect the degree of instruction-level parallelism achieved.

In order to attenuate the aforementioned stalls, hardware vendors have devised several techniques. For example, to minimize memory-related stalls, smaller and faster *cache* memories are placed in the data path between processor and main memory. The strategy is to benefit from the locality principle and serve most memory requests with data coming from lower-latency cache accesses. Additionally, data dependencies are minimized by allowing instructions to execute out-of-order inside the pipeline. Finally, control dependencies are partially addressed via speculative execution (i.e., the processor executes instructions that lie beyond a conditional branch as if it had been already resolved).

In practice, the characteristics of the applications determine whether the hardware techniques above will be successful or not at making the processor execute close to its full capacity. With that in mind, a number of novel analytical databases had been developed over the last years, in an attempt to better exploit the internal features of processors. Examples of microarchitectural optimizations employed by such databases include a column-oriented data organization and compression techniques which together provide a more efficient use of memory hierarchy [12, 13]. Also, recently proposed compression algorithms [14] minimize the negative impact of branch mispredictions by removing `if-then-else` constructs from their critical path.

We argue that similar optimizations can be applied in the context of event processing systems, resulting in potentially higher gains since in this case data is manipulated mostly in main memory.

3 Assessing and Improving CPU Performance of EP Systems

This section introduces the workload, benchmark setup and an evaluation of Esper and of our own prototype with and without CPU and cache-aware optimizations.

3.1 Workload

In order to assess the gains of the optimizations proposed in this work we used a simple, though representative workload, composed by two common operations performed by event processing systems: moving aggregations over event streams and correlation of event streams with historic data (*join*). These two queries and the dataset are described in detail next.

Dataset
Input data consisted in a generic event stream *S* and table *T* with schemas shown in Figure 1 and Figure 2.

```
CREATE STREAM S (
    ID integer,
    A1 double,
    A2 double,
    TS long)
```

Fig. 1. Schema of input dataset (stream "S")

In our tests the values assumed by `S` attributes are not relevant for query performance evaluation, so they were filled with a fixed, pre-generated, data value – *this ensures that measurements are minimally affected by data generation*. The exception is the attribute `ID`, which is used to join stream `S` with table `T`. In this case, `S`'s `ID` was filled with random values uniformly distributed in the range of `T`'s `ID`.

```
CREATE TABLE T (
    ID integer,
    T1 integer,
    T2 integer,
    T3 integer,
    T4 integer)
```

Fig. 2. Schema of table "T" used in the join query

`T`'s `ID` attribute, used to perform the join with the event stream, assumes unique values ranging from 1 to `TABLE_SIZE_IN_ROWS`. The other four attributes in `T` do not influence query performance and are filled with random data.

Aggregation Query
The aggregation query used in this study computes a moving average over a *count-based sliding* window[1]. The query is shown below using syntax of CQL [4], a SQL-based query language for continuous stream processing:

```
SELECT avg(A1)
FROM S [ROWS N Slide 1]
```

Fig. 3. Aggregation query, written in CQL

For this particular query, every event arrival at stream `S` causes the output of an updated result. Parameter `N` represents the window size, which varies across the tests, ranging from 1000 to 100 million events.

[1] "*Count-based*" means that the window size is defined in terms of the number of events over which the computation takes place. "*Sliding*" means that once the window is full events are expired one-by-one upon arrival of new events.

Join Query

To examine the behavior of an event processing system when performing a join, we used a query based on a real use-case of a telecom company that needed to join streaming call detail records (CDR) (here represented by stream `S`) with historic data (represented by table `T`). In our tests this query is expressed as follows:

```
SELECT S.ID, S.A1, T.T1
FROM   S, T
WHERE  S.ID = T.ID
```

Fig. 4. Join query – stream S with table T

Since the goal here is to focus on the performance of processor and memory hierarchy, the table is maintained in main memory, thus eliminating eventual effects of the I/O subsystem on the results. (*As pointed out before, keeping the dataset in memory is commonplace in most EP applications, especially those which require high processing throughputs and/or low latencies.*) The selectivity of the query is always 100% (i.e., every event is matched against one and only one record in the table) and the table size is varied across tests, ranging from 1 thousand to 10 million rows.

3.2 Experiments: Setup and Methodology

All the tests were carried out on a server with two Intel Xeon E5420 Quad-Core processors (Core® microarchitecture, L2-Cache: 12MB, 2.50 GHz, 1333 MHz FSB), 16 GB of RAM, running Windows Server 2008 x64 and Sun Hotspot x64 JVM.

The performance measurements were done as follows:

- A single Java application was responsible for generating, submitting and consuming tuples during the performance runs. Events are submitted and processed through local method calls, so that measurements are not affected by network/communication effects.
- In the tests with join queries, load generation was preceded by an initial loading phase, during which the in-memory table was populated with a given number of records.
- Load generation started with an initial 1-minute warmup phase, with events (`S` tuples) being generated and consumed at the maximum rate supported.
- Warmup was followed by a 15-minute measurement phase, during which we collected both application-level metrics and hardware-level metrics. Application-level metrics, namely throughput and memory consumption, were gathered inside the Java application. Throughput was computed as total event count divided by elapsed time. Memory consumption was computed using `totalMemory()` and `freeMemory()`, standard Java SDK `Runtime` class methods. Hardware-level metrics were obtained using the *Intel VTune*® profiler [8], as described in detail next. As in the warmup phase, events were submitted at the maximum rate sustained by the specific implementation.
- Each test was repeated 3 times and the reported metrics were averaged.

VTune collects metrics by inspecting specific hardware counters provided by Intel processors. To avoid excessive monitoring overhead, only one or two events are collected at a time – for doing that, VTune breaks event collection in "runs". In our tests, each run has a duration of 2 minutes (1 minute for calibration and 1 minute for counters collection), and 6 runs were necessary to collect all the configured metrics.

We started the experimental evaluation by measuring the performance of the two queries in the open-source EP engine Esper (Section 3.3) and then implemented the same basis algorithms employed by it on a separate, custom-code, Java prototype. These baseline algorithms were then progressively modified to include the proposed optimizations while we performed new measurements[2] (Sections 3.4 and 3.5). Finally, we validated the optimizations by measuring their performance in more realistic scenarios: first with the prototype processing several queries simultaneously (Section 3.6) and then with the optimizations being inserted into Esper code (Section 3.7).

The basis algorithm for computing the aggregation query works as follows. The *sliding window* is implemented as a circular buffer, internally represented as a fixed-length array; the average aggregation itself is computed by updating count and sum state variables upon event arrival/expiration. The join algorithm is also straightforward: it keeps the table into a hash index structure with the join attribute as key and then performs a lookup in that hash table every time a new event arrives.

3.3 Preliminary Results

Figure 5 illustrates the results for the join query running at the event processing engine Esper, using two different tuple representations: *Map* and *POJO* (in the former events are represented as instances of the standard `HashMap` Java class, while in the latter events are represented as fixed-schema Plain Java Objects)[3]:

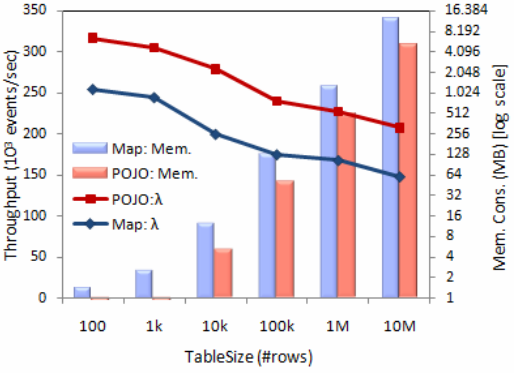

Fig. 5. Performance of join query on Esper

Two important observations can be made from the graph above: First, for both tuple representations, the throughput dropped about 40 percent from a join with a 1000-rows table to a join with a 10M-rows table. This drop occurred even though the employed algorithm – hash join – has a O(1) runtime

[2] We could have implemented the changes only on Esper, but our goal was to assess the performance gains of the optimizations on query processing in spite of any product specificities. Besides, using a smaller custom-code application makes the effects of code changes more controllable and understandable.

[3] In this work, we focus on Esper due to its open-source nature, which allowed us to port some of the optimizations here proposed into a real engine. We point out, however, that the results here presented are representative of a range of event processing engines, given that the behavior exhibited by Esper is similar to what was observed in previous tests with other products (see [10] for more details).

complexity. Second, measured memory consumption during query execution was 4 to 60 times the space required for keeping the table data. Examining carefully the hardware metrics and Esper source code, we concluded that the drop on query performance was due mainly to microarchitectural aspects (and to a lesser extent, to increased garbage collection activity) and that the excessive memory consumption was caused by non-optimized internal representation of tuples as further explored in the next section.

3.4 Optimizing Internal Data Structures

In order to address the problem of excessive memory consumption, we focused first in optimizing the structures used to keep data items in main memory (i.e., the window for the aggregation query, and the table for the join query). Specifically, we were interested in finding out if the original representations used by Esper to represent the stream and table tuples could be improved and if a column-oriented storage model would result in enhanced performance in the context of event processing.

It is worthy to notice that column-store formats are especially useful for read-oriented, scan-oriented, non-ad-hoc queries. Thus, while on one hand EP systems, with their scan-oriented, non-ad-hoc queries may benefit from column-store representations, on the other hand, EP systems read/write workloads might be hurt by column-store representations.

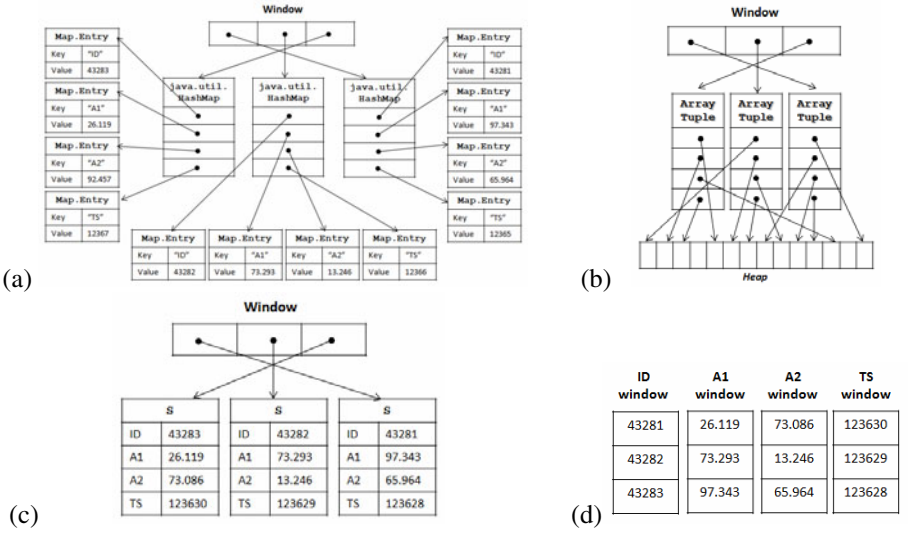

Fig. 6. The different data structures used to represent tuples: (a) key-value Map; (b) array of Objects; (c) Plain Object (POJO); (d) Column-Store

To assess the impact of data structures on query performance, we started representing events/tuples as instances of the `HashMap` class – Figure 6(a) – and then employed progressively more lightweight representations: first as arrays of Objects (b), and then as fixed-schema Plain Java Objects (POJO) (c). Finally we tested the column-oriented storage model (d), in two different modalities: first keeping all original attributes of events/tuples (here named "*Col-Store*") and then keeping (projecting)

only the attribute referenced in the query ("*Col-Store Proj.*"). In the *Col-Store* format, N aligned arrays of primitive types are kept in memory (where "N" is the number of attributes), while in the *Col-Store Proj* format only one array containing the values for the referenced attribute is maintained. Figure 7, Table 1 and Table 2 summarize the most relevant results for both the aggregation and join queries:

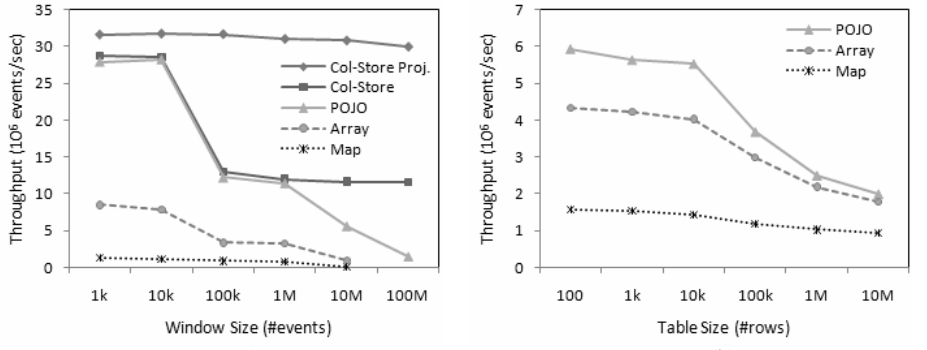

Fig. 7. Impact of internal representation on performance: (a) aggregation; (b) join[4].

Table 1. Data structures and memory consumption (in MB): aggregation query

Tuple Format	Window Size					
	1k	10k	100k	1M	10M	100M
Map	0.8	5.5	51.4	511.5	5,112.9	-
Array	0.5	2.1	17.9	176.5	1,755.0	-
POJO	0.3	0.8	6.3	61.2	682.1	6,103.0
Col-Store	0.2	0.5	2.8	26.9	267.2	2,670.5
Col-Store Proj.	0.2	0.3	0.9	7.8	76.0	763.1

Table 2. Data structures and memory consumption (in MB): join query

Tuple Format	Table Size					
	100	1k	10k	100k	1M	10M
Map	0.2	1.1	9.3	92.1	920.0	9,131.2
Array	0.2	0.4	2.8	27.3	268.4	2,654.8
POJO	0.2	0.3	1.4	13.6	130.6	1,275.3
Col-Store	0.2	0.3	1.4	13.2	126.8	1,237.0
Col-Store Proj.	0.2	0.3	1.3	11.7	111.5	1,083.8

[4] For the sake of clarity, the lines "*Col-Store*" and "*Col-Store Proj*" were omitted in the Join graph since the results with these two implementations were very similar to the POJO case.

The numbers reveal that considerable gains in final performance and resource consumption could be obtained by using more optimized data structures. For the join query, the most efficient representation achieved more than 3 times more throughput than the original one, while consuming less than 15% memory space of what was originally required. The gains for the aggregation query were even more impressive: the best performing implementation (column-store keeping only required data) achieved on average about 35 times more throughput than the one using maps as event representation and reduced memory consumption to around 1,5% of the memory space occupied by this initial implementation.

Discussion: Aggregation Query

A couple of facts in the aggregation tests results are worth mentioning. First, using optimized data structures allowed the aggregation query to operate over windows of larger sizes, which otherwise would not be possible if employing the original non-optimized tuple representations (e.g., it was not possible to run the aggregation query over a window of 100M events when they were represented as Maps or arrays of Objects because these implementations required more memory space than it was physically available).

Second, the POJO format, although more efficient than Map and array representations, suffered severe drops in performance in two distinct points of the graph: from 10k to 100k and from 1M on. The collected metrics reveal that the first drop was caused by microarchitectural aspects (more specifically, an increase in L2 cache misses), while the second was due to an increased garbage collection activity.

Further results indicate that the column-oriented storage model addressed partially or even totally the aforementioned issues. For instance, in contrast with the *POJO* line, the *Col-Store* line in Figure 7(a) remained steady after 1M. The reason for such difference is that the column-oriented representation, by using primitive types instead of Objects, is less susceptible to garbage collection than the POJO. In the best case (i.e., when keeping in memory only the attribute referenced by the query), the column-oriented model was also able to eliminate the microarchitectural issues. For that particular implementation, the memory access pattern is essentially sequential – consuming events from the stream and inserting them into the window means sequentially traversing a primitive type array – which maximizes performance at the microarchitectural level and ensures a steady throughput over the most different window sizes. Indeed, this observation was corroborated experimentally, with the CPI metric remaining basically unaffected in all tests of that specific implementation (ranged from 0.850 to 0.878 for windows of 1000 up to 100M events).

Discussion: Join Query

Interestingly, for the join query the column-oriented storage model did not provide considerable performance gains with respect to the POJO representation. (In fact, both throughput and memory consumption were quite similar in these two configurations and for this reason we omitted the results for column-store in Figure 7(b)) This behavior seems to be related to the fact that the amount of memory consumed by the payload itself (i.e., tuples attributes) in that case is small compared to the overhead of inherent factors of the Java programming environment (e.g., object alignment,

wrappers for primitive types, and especially heavyweight native `HashMap` class, the structure used for indexing the table), which hides eventual gains provided by a more concise column-oriented dataset. Overall, the final performance of column-store implementation for the join query oscillated around +3% and -5% in comparison with the tests with POJO tuples.

3.5 Improving Algorithms Efficiency at the CPU Level

Nearly all results presented in previous section revealed that the throughput of both aggregation and join queries dropped significantly as the input size was increased, even though the theoretical runtime complexity of the employed algorithms is O(1). In this section we delve into the causes for this behavior and propose optimizations to improve algorithms scalability with respect to input size.

Aggregation Query

As mentioned before, the algorithm for computing *subtractable* [5] aggregations such as `AVG`, `SUM` or `COUNT` over sliding windows consists essentially in updating some fixed set of state variables upon event arrival while maintaining the events of the window in main memory. As such, the algorithm has a O(N) space complexity but a theoretical O(1) time complexity. In practice, however, several factors can make the running time of the algorithm grow when the input size is increased. One of them is garbage collection: generally, the bigger the working set size and the higher the throughput, the more time will be spent on GCs. Besides, execution efficiency at the CPU will typically be hurt when more elements are referenced due to an increased probability of cache misses. This is particularly the case when events are represented as Objects, because there is no guarantee that consecutive elements in the window will be allocated contiguously in the heap by the JVM. Therefore, even though the algorithm logically traverses the window in a sequential way, the memory access pattern tends to be essentially random. One possible way of eliminating this undesirable effect is to employ the column-oriented storage model, which avoids the random walks through the heap by keeping attributes as arrays of primitive types. However, this approach is worthwhile only if the number of attributes referenced in the query is small. Otherwise, consuming an event from the stream and inserting it into the window will involve accessing several memory locations (one entry per attribute in different arrays).

We tested, then, a tuned algorithm to minimize this performance penalty due to multiple inserts. The idea is to avoid references to distant memory locations by using a L2-resident temporary buffer for accommodating the incoming events. This temporary buffer consists in N aligned arrays (one per attribute) as in the original window, but with a capacity of only 100 events. Once these small arrays get full, the events are copied back to the window, one attribute at a time, so that they can be expired later.

The algorithm is described in detail in Table 3. Figure 8 compares the performance of the proposed algorithm with the original column-oriented implementation.

Table 3. Cache-Aware algorithm for computing aggregations over sliding windows

Input: S: incoming event stream
K: the size of the L2-resident temporary buffer
Output: R: stream of results

for each event E in S **do**
  for each attribute A_i of event E **do**
    store A_i on the corresponding temporary location T_i
  compute aggregation (update aggregator state)
  insert aggregation result into output stream R
  if temporary buffer T is full **then**
    for each attribute A_i of event E **do**
      for each item I_j in temporary buffer T_i **do**
        copy I_j to the appropriate location in corresponding window W_i
    reset the temporary location T
    slide the window W in K positions

The optimized algorithm provided gains in performance that ranged from 28 to 35 percent when compared to the original column-store. The hardware metrics confirmed that it indeed exploits better the characteristics of the CPU: the CPI was almost half of the CPI of the original column-store and L2 cache miss rate was reduced to around 70% of what was originally measured. Evidently, this microarchitecture-aware algorithm is best-suited for medium-to-large windows, since for smaller sizes the working set of the original column-oriented implementation already fits in L2 cache.

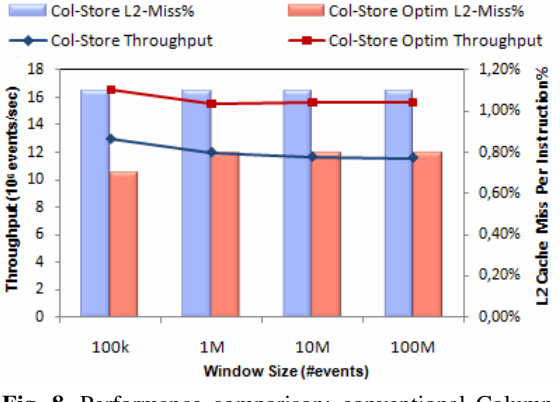

Fig. 8. Performance comparison: conventional Column-Store algorithm vs. Microarchitecture-Aware algorithm

Join Query

In theory, the cost of a lookup on a hash table should be independent on the number of elements stored on it, but this ideal rarely can be achieved in practice since CPU operation – specifically cache behavior – is considerably affected by working set size. Figure 9 illustrates the correlation between microarchitectural aspects and final system performance for the join tests with tuples represented as POJO.

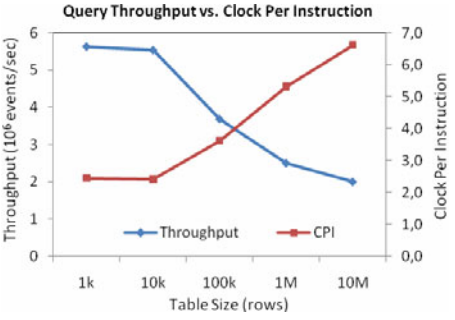

Fig. 9. Drop on join performance due to increase in CPI

As it can be noticed from the graph, query throughput falls in the same proportion as the clock per instruction metric increases, indicating that microarchitectural factors were the major cause for performance degradation when the table size was increased. Further analysis of the hardware metrics confirmed that the increase in CPI was due to less efficient memory access patterns. For example, as the table size was increased from 1,000 rows to 10M rows, *L2 cache miss per instruction* metric went from 0.6% to 3.1% and *TLB miss penalty* metric jumped from 0.3% to 19.4%. Notice also that up to 10k rows, the table fits in the 12MB L2 cache, which explains the negligible performance drop from 1k to 10k, and the significant degradation from that point on.

To improve data locality, we implemented an adapted version of the grace hash join algorithm [9] used in DBMSs. The idea is to reduce the number of times data is brought from main memory to cache by splitting the whole table into partitions and accessing them in bulks. The algorithm works as follows:

- When the table is being populated, the records are stored into partitions using a given partitioning function *g* (in our tests the table was split into 1000 partitions);
- Incoming events are then processed in batches. They are buffered into a partitioned list until the batch is complete. (The partitioning function is the same as the one used for splitting the table, which ensures that matching tuples in the batch and the table will be kept in corresponding partitions).
- Once the batch is complete, the corresponding partitions from event batch and table are loaded in pairs. The event partitions are then sequentially scanned, performing for every event a lookup on the corresponding table partition.

Figure 10 shows test results with both the conventional hash join algorithm and the batch grace hash algorithm, for table sizes ranging from 10M rows to 80M rows.

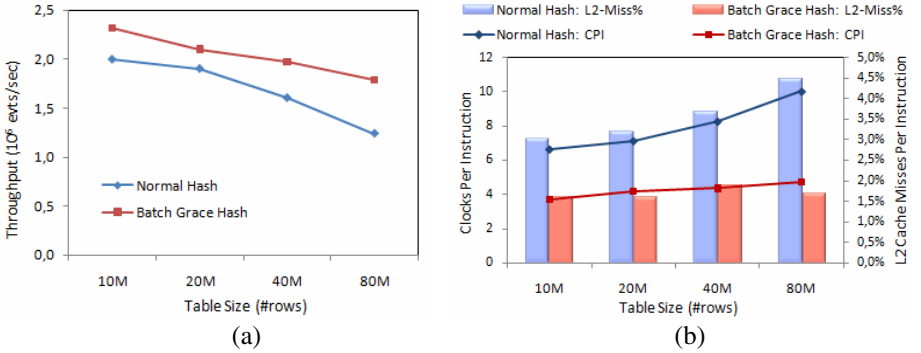

Fig. 10. Conventional Hash Join vs. Batch Grace Hash Join

As it can be seen in (b), the batch algorithm was successful in improving locality of data accesses, which in turn caused a reduction in average CPI. This resulted in performance gains that ranged from 11 to 44 percent, as illustrated in (a).

Notice that there are a couple of competing factors influencing the performance of the batch grace hash algorithm. For example, each table partition should ideally fit in L2 cache in order to minimize the high penalties associated with memory accesses. Assuming that table size is application-specific and cannot be changed, the only way this can be achieved is by increasing the number of partitions in the table. Doing so, however, means that the number of partitions in the batch of events is also increased, thus reducing the number of events per partition. A reduced number of events per partition will probably hurt performance as a good fraction of the lookups in the table will incur in compulsive cache misses. For avoiding this to happen, batch size could be increased in the same proportion as the number of partitions, but obviously this is only feasible if there is availability of memory resources. Determining the optimal values for these parameters is subject for further investigation.

3.6 Optimizations in a Multi-query Scenario

A natural question that might arise after analyzing the results of previous sections is whether similar improvements in performance would be observed when moving from a scenario with only one continuous query running at a time to a multi-query scenario. In this section we answer this question and present the results for a set of tests in which the proposed optimizations are validated by measuring system performance during the execution of multiple simultaneous queries. More specifically, we tested three different situations:

i. N instances of the same query are computed over a *single event stream* in a *single thread*;
ii. N instances of the same query are computed over *independent but identical event streams* in a *single thread*;
iii. N instances of the same query are computed over *independent but identical event streams* in *N threads*.

For aggregation, we performed tests with 1 up to 16 simultaneous queries, with sliding windows of 10 million events each. For join queries, we tested 1 up to 8 simultaneous queries operating over N tables of 10 million records (there was no physical memory available to test with 16 queries). We then analyzed the evolution of throughput and hardware-level metrics as we progressively added more queries to the configuration. The output throughput of each setting is shown in Figure 11.

Application-level and hardware-level metrics collected during tests indicate that the proposed optimizations are also effective in a multiquery scenario. For instance, the microachitecture-aware aggregation algorithm introduced in Section 3.5 achieved superior performance than the conventional column-store in all multi-query tests. Also, the "*Col-Store Proj*", which achieved the highest throughputs in the single aggregation query scenario due to improved microarchitectural performance, was once more the best performing implementation.

For join, the speedup of the batch grace hash algorithm over the conventional hash algorithm oscillated between 1.02 to 1.64. On average, the optimized implementation

achieved 22 percent more throughput than the conventional non-optimized one, a slightly better speedup than the one observed in the single query scenario (20 percent). Once more, hardware-level metrics followed the trend observed in the single query scenario (on average, the conventional hash algorithm had a CPI of 7.52 and L2 cache miss rate of 3.5% against a CPI of 3.46 and a L2 cache miss rate of 1.6% for the batch grace hash algorithm).

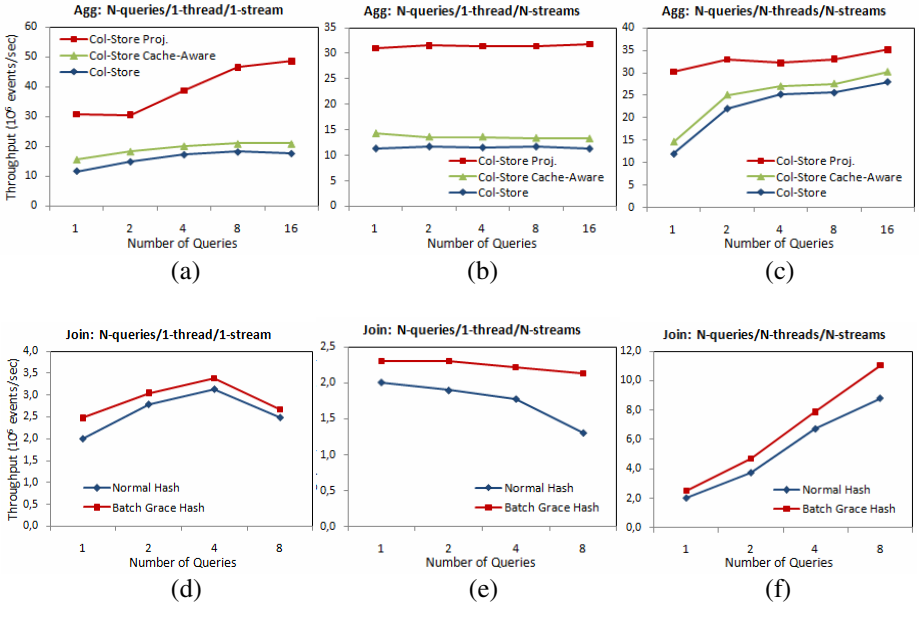

Fig. 11. Output throughput for aggregation and join queries in multi-query scenario

A few words about the shape of the curves: on (a) the output throughput incresased sligthly when more queries were added as a result of a reduction on the relative weight of event instantiation on the workload (event is created once, but processed N times). This constrasts with the workload on (b) where event instantiation is replicated in the same proportion as queries, which explains the steady line (individual throughput of queries decreases, but total system throughput remains the same). Interestingly, the curves on (c) did not present a linear (*or close to linear*) growth that one would expect when increasing the amount of resources for a set of independent tasks (as happened on (f), for example). We found out that this happened because the CPI metric in these tests increased essentially in the same proportion as the number of queries. The reason for this behavior, however, is unclear to us since the other hardware metrics (i.e., cache miss rates, instruction fetch stalls, resource stalls, etc.) did not show any change that could justify this increase in CPI. Finally, it should be noticed that the drop on performance when jumping from 4 to 8 queries on (d) was caused by increased garbage collection activity (for 8 queries the system ran close to the maximum memory available).

3.7 Optimizations on a Real EP System

In this section we discuss the results of the tests performed with the event processing engine Esper, after implementing the proposed optimizations. Figure 12 below shows the most relevant findings.

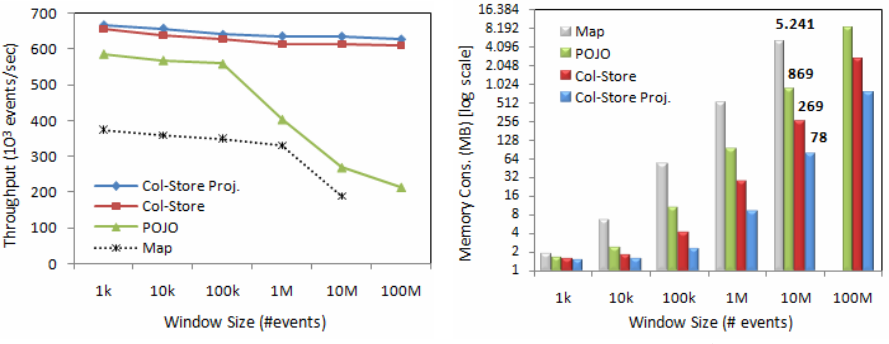

Fig. 12. Optimizations on Esper (aggregation query): (a) Throughput (b) Memory Consumption

As in the tests with the EP prototype, the modified version of Esper using a column-oriented storage model achieved higher throughputs and scaled better than the original implementations using Maps or POJOs as event representation. The column-oriented implementations also proved once more to be useful for reducing memory consumption for aggregation queries.

It should be noticed however, that many of the performance gains obtained with other optimizations in the tests with the prototype were hidden by bottlenecks inherent to Esper implementation. This phenomenon can be seen in Figure 12(a), where the "*Col-Store Proj.*" implementation achieved only 2,7% more throughput than the conventional "*Col-Store*", while on the EP prototype the same optimization provided twice more performance. For the same reason, the cache-aware algorithm for aggregation and the batch grace hash join algorithm provided only slight performance gains on Esper (around 1 percent).

4 Conclusions and Future Work

In this paper we started the discussion on how event processing systems can execute more efficiently at processors of today. We first collected application and hardware-level metrics during execution of a real event processing system to determine its performance at CPU. After identifying some issues, we proposed, implemented and evaluated some changes in data organization and algorithms that together provided more than order of magnitude performance gains and considerable reductions on memory consumption. We have also found out that:

1. Microarchitectural aspects played a fundamental role on the performance degradation observed when input size was increased;

2. The column-oriented storage model can greatly improve performance for aggregation queries when compared to alternative representations like Plain Java Objects or key-value Maps. These performance gains, whose roots are in an improved microarchitectural execution as well as in a reduced garbage collection activity, are also followed by significant memory savings (56% up to 98% reduction);
3. The proposed cache-aware optimizations remained effective in a scenario with multiple queries running simultaneously.

As future directions, we plan focus in the development of more optimized structures for join queries as well as to work in the optimization of other classes of continuous queries (e.g., event pattern detection). Other interesting topics worth of further investigation include how to make a better use of the parallelism provided by modern multicore processors and the compromise between CPU-level optimizations and query-plan sharing.

References

1. Abadi, D.J., Madden, S.R., Hachem, N.: Column-stores vs. row-stores: how different are they really? In: Proc. of the 2008 ACM SIGMOD Vancouver, Canada (2008)
2. Ailamaki, A., DeWitt, D.J., Hill, M.D., Wood, D.A.: DBMSs On A Modern Processor: Where Does Time Go? In: Proc. of the 25th VLDB Conference, Edinburgh, Scotland (1999)
3. Ailamaki, A., De Witt, D.J., Hill, M.D., Wood, D.A., Skounakis, M.: Weaving Relations for Cache Performance. In: Proc.of the 27th VLDB Conference, Roma, Italy (2001)
4. Arasu, A., Babu, S., Widom, J.: The CQL continuous query language: semantic foundations and query execution. VLDB Journal 15(2), 121–142 (2006)
5. Arasu, A., Widom, J.: Resource Sharing in Continuous Sliding-Window Aggregates. In: Proceedings of the 30th VLDB Conference, Toronto, Canada (2004)
6. Esper, `http://esper.codehaus.org/`
7. Hennessy, J.L., Patterson, D.A.: Computer Architecture: A Quantitative Approach, 4th edn. Morgan Kaufman Publishers, San Francisco (2007)
8. Intel VTune, `http://software.intel.com/en-us/intel-vtune/`
9. Kitsuregawa, M., Tanaka, H., Moto-Oka, T.: Application of Hash to Data Base Machine and Its Architecture. New Generation Computing 1(1), 63–74 (1983)
10. Mendes, M.R.N., Bizarro, P., Marques, P.: A performance study of event processing systems. In: Nambiar, R., Poess, M. (eds.) TPCTC 2009. LNCS, vol. 5895, pp. 221–236. Springer, Heidelberg (2009)
11. Ramamurthy, R., DeWitt, D.J.: Buffer-pool Aware Query Optimization. In: CIDR 2005, pp. 250–261 (2005)
12. Stonebraker, M., Abadi, D.J., et al.: C-Store: A Column-Oriented DBMS. In: Proc. of the 31st VLDB Conference, Trondheim, Norway (2005)
13. Zukowski, M., Boncz, P.A., Nes, N., Heman, S.: MonetDB/X100 - A DBMS In The CPU Cache. IEEE Data Engineering Bulletin 28(2), 17–22 (2005)
14. Zukowski, M., Heman, S., Nes, N., Boncz, P.A.: Super-Scalar RAM-CPU Cache Compression. In: Proc. of the 22nd ICDE, Atlanta, USA (2006)

Author Index