BDMS Performance Evaluation: Practices, Pitfalls, and Possibilities

Michael J. Carey

Information Systems Group, Computer Sciences Department,
University of California, Irvine,
Irvine, CA 92697-3435
mjcarey@ics.uci.edu

Abstract. Much of the IT world today is buzzing about Big Data, and we are witnessing the emergence of a new generation of data-oriented platforms aimed at storing and processing all of the anticipated Big Data. The current generation of Big Data Management Systems (BDMSs) can largely be divided into two kinds of platforms: systems for Big Data analytics, which today tend to be batch-oriented and based on MapReduce (*e.g.,* Hadoop), and systems for Big Data storage and front-end request-serving, which are usually based on key-value (*a.k.a.* NoSQL) stores. In this paper we ponder the problem of evaluating the performance of such systems. After taking a brief historical look at Big Data management and DBMS benchmarking, we begin our pondering of BDMS performance evaluation by reviewing several key recent efforts to measure and compare the performance of BDMSs. Next we discuss a series of potential pitfalls that such evaluation efforts should watch out for, pitfalls mostly based on the author's own experiences with past benchmarking efforts. Finally, we close by discussing some of the unmet needs and future possibilities with regard to BDMS performance characterization and assessment efforts.

Keywords: Data-intensive computing, Big Data, performance, benchmarking, MapReduce, Hadoop, key-value stores, NoSQL systems.

1 Introduction (The Plan)

We have entered the "Big Data" era – an era where a wealth of digital information is being generated every day. If this information can be captured, persisted, queried, and aggregated effectively, it holds great potential value for a variety of purposes. Data warehouses were largely an enterprise phenomenon in the past, with large enterprises being unique in recording their day-to-day operations in databases and warehousing and analyzing historical data in order to improve their business operations. Today, organizations and researchers from a wide range of domains recognize that there is tremendous value and insight to be gained by warehousing the emerging wealth of digital information and making it available for querying, analysis, and other purposes. Online businesses of all shapes and sizes track their customers' purchases, product searches, web site interactions, and other information to increase the effectiveness of their marketing and customer service efforts; governments and businesses track the content of blogs and tweets to perform sentiment analysis; public health organizations monitor news articles, tweets, and web search trends to track the progress of epidemics; and, social scientists study tweets and social networks to understand how information of various kinds spreads and how it can be effectively utilized for the public good. Technologies for data-intensive computing, search, and scalable information storage – *a.k.a.* Big Data analytics and management – are critical components in today's computing landscape. Evaluating and driving improvements in the performance of these technologies is therefore critical as well.

The goal of this paper is to take an informal look, with a critical eye, at the current state of the art in Big Data platform performance evaluation. The eye in question will be that of the author, who makes no claims about being an actual expert in this area. The author’s performance evaluation experience comes mostly from a series of previous forays into benchmarking of other database technologies, and his Big Data experience comes from a current and somewhat counter-cultural project (ASTERIX) that aims to develop a second-generation (meaning post-Hadoop) Big Data Management System (BDMS) at UC Irvine. The paper will start by reviewing some of the history in the previously distinct areas of Big Data technologies and DBMS benchmarking; this part of the paper will end with a summary of where things are today at the intersection of these two areas. The paper will then turn to a series of potential pitfalls – things to be wary of – when attempting to characterize and/or to compare the performance of data management systems; this part of the paper will largely be anecdotal, drawing on lessons that the author has learned either by direct observation or through personal experience. The paper will then turn briefly to the question of future requirements and challenges, presenting one perspective on where future efforts in this area might want to focus; this part of the paper will be based largely on combining inputs that the author has gotten from various industrial colleagues together with some of the lessons covered in the middle part of the paper.

2 Background (The Practices)

In this section of the paper we will take quick tours of the history of systems for managing Big Data, of some of the historical efforts to benchmark data management technologies, and of the current state of these two fields (combined).

2.1 Big Data Management Systems

The IT world has been facing Big Data challenges for over four decades, though the meaning of “Big” has obviously been evolving. In the 1970’s, “Big” meant Megabytes of data; over time, “Big” grew first to Gigabytes and then to Terabytes. Nowadays the meaning of “Big” for data in the enterprise IT world has reached the Petabyte range for high-end data warehouses, and it is very likely that Exabyte-sized warehouses are lurking around the corner.

In the world of relational database systems, the need to scale to data volumes beyond the storage and/or processing capabilities of a single large computer system gave birth to shared-nothing parallel database systems [23]. These systems run on networked clusters of computers, each with their own processors, memories, and disks. Data is spread over the cluster based on a partitioning strategy – usually hash partitioning, but sometimes range partitioning or random partitioning – and queries are processed by employing parallel, hash-based divide-and-conquer techniques. The first generation of systems appeared in the 1980's, with pioneering prototypes from the University of Wisconsin and the University of Tokyo, a first commercial offering from Teradata Corporation, and traditional relational DBMS vendors following suit. The past decade has seen the emergence of a new wave of systems, with a number of startups developing parallel database systems that have been swallowed up through recent acquisitions by IBM, Microsoft, EMC, HP, and even Teradata. Users of parallel database systems have been shielded from the complexities of parallel programming by the provision of SQL as a set-oriented, declarative API. Until quite recently, shared-nothing parallel database systems have been the single most successful utilization of parallel computing, at least in the commercial sector.

In the late 1990's, while the database research community was admiring its "finished" research on parallel databases, and the major database software vendors were commercializing the results, the distributed systems world began facing Big Data challenges of its own. The rapid growth of the World-Wide Web, and the resulting need to index and query its mushrooming content, created Big Data challenges for search companies like Inktomi, Yahoo!, and Google. Their processing needs were quite different, so parallel databases were not the answer, though shared-nothing clusters emerged as the hardware platform of choice in this world as well. Google responded to these new challenges [21] by developing the Google File System (GFS), allowing very large files to be randomly partitioned over hundreds or even thousands of nodes in a cluster, and by coupling GFS with a very simple programming model, MapReduce, that enables programmers to process Big Data files by writing two user-defined functions, map and reduce. The Google MapReduce framework applied these functions in parallel to individual data items in GFS files (map) and then to sorted groups of items that share a common key (reduce) – much like the partitioned parallelism used in shared-nothing parallel database systems. Yahoo! and other big Web companies such as Facebook soon created an open-source version of Google's Big Data stack, yielding the now highly popular Apache Hadoop platform [4] and its associated HDFS storage layer. Microsoft has a different but analogous Big Data stack, the SCOPE stack [19], used in support of its Bing search services.

Similar to the two-worlds history for Big Data back-end warehousing and analysis, the historical record for Big Data also has a dual front-end (*i.e.,* user-facing) story worth noting. As enterprises in the 1980's and 1990's began to automate more and more of their day-to-day operations using databases, the database world had to scale up its online transaction processing (OLTP) systems as well as its data warehouses. Companies like Tandem Computers responded with fault-tolerant, cluster-based SQL systems. Similarly, but again later over in the distributed systems world, the big Web companies found themselves driven by very large user bases (up to 10s or even 100s of millions of Web users) to find solutions to achieve very fast simple lookups and updates to large, keyed data sets such as collections of user profiles. Monolithic SQL databases for OLTP were rejected as too expensive, too complex, and not fast enough, and the scalable key-value stores that are driving today's "NoSQL movement" [18] were born. Again, companies like Google and Amazon each developed their own answers (BigTable and Dynamo, respectively) to meet this set of needs, and the Apache community soon followed suit by creating a set of corresponding open-source clones (*e.g.,* HBase and Cassandra).

So where are things now? Over the past few years, Hadoop and HDFS have grown to become the dominant platform for Big Data analytics at large Web companies as well as within less traditional corners of traditional enterprises (*e.g.,* for click-stream and log analyses). At the same time, data analysts have grown tired of the low-level MapReduce programming model; instead, they are now using a handful of high-level declarative languages and frameworks that allow data analyses to be expressed much more easily and written and debugged much more quickly. The two most popular languages are Hive [5] from Facebook (a variant of SQL) and Pig [6] from Yahoo! (a functional variant of the relational algebra, roughly). Tasks are first expressed in these languages and then compiled into a series of MapReduce jobs for execution on Hadoop clusters. Looking at the workloads on real clusters in the last few years, it has been reported that well over 60% of Yahoo!'s Hadoop jobs and over 90% of Facebook's Hadoop jobs come from these higher-level languages rather than hand-written MapReduce jobs. More and more, MapReduce is being relegated to serving as the Big Data runtime for various higher-level, declarative data languages (which are not so different from SQL) rather than as a solution developers' programming platform. Similarly, and ironically, there are even early efforts looking at providing higher-level, SQL-like query language interfaces to "NoSQL" stores. Figure 1 illustrates the layers in a typical instance of the first-generation, Apache-based Big Data software stack.

The first-generation Hadoop-centric software stack for Big Data dominates today in industry, and batch-oriented data analytics are usually managed separately (both

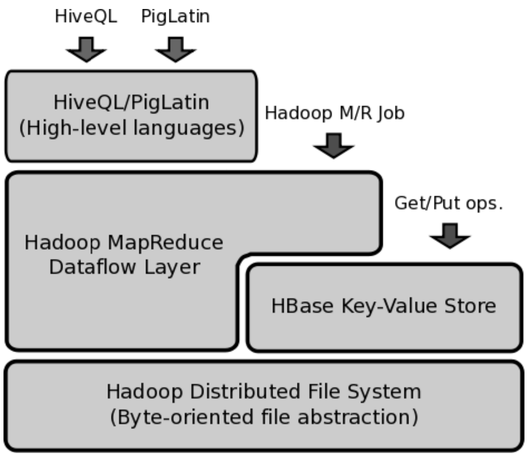
HiveQL
PigLatin
HiveQL/PigLatin (High-level languages)
Hadoop M/R Job
Hadoop MapReduce Dataflow Layer
Get/Put ops.
HBase Key-Value Store
Hadoop Distributed File System (Byte-oriented file abstraction)

Fig. 1. The first-generation, Hadoop-based, Big Data software stack

logically and physically) from the real-time, user-facing, key-value stores in today's overall architectures. ETL-like processes, usually also Hadoop-based, are used to connect the two. In addition to these current architectures, it is important to be aware, when planning Big Data performance studies, that changes are occurring in this space and that performance-related efforts must be ready for the changes. One such trend is towards more specialized systems on the Big Data analytics side. As examples, the increasing availability of very large graph data sets, such as social graphs or derived graphs of user interactions, is leading to the creation of new platforms such as Pregel [27] and GraphLab [26] for graph analytics or programming models; a new platform called SciDB [33] is currently being developed for storing, querying, and analyzing large volumes of array-based science data; and, the development of platforms tailored to large-scale machine learning tasks (*e.g.,* see [12]) is becoming a popular target driven by Big Data analysis requirements. Another potential "trend" is represented by the ASTERIX project at UC Irvine [8], where we are working to deliver a BDMS that is somewhat *less* specialized – one where "one size fits a bunch" – namely, a system that is capable of handling very large quantities of semistructured data and supporting data ingestion, updates, small to medium queries, and large batch analyses over both internally managed as well as externally stored data [9, 11, 1].

2.2 Data Management Benchmarks

Benchmarking of database management systems [24] is an activity that has drawn the attention of DBMS practitioners and researchers for over three decades. Two of the most influential early benchmarks were the Wisconsin benchmark and the Debit-Credit benchmark.

The Wisconsin benchmark [22] was developed in the early 1980's and was the first benchmark designed to test and compare the performance of relational database systems. The benchmark was a single-user micro-benchmark consisting of 32 queries chosen to measure the performance of basic relational operations. The benchmark query set included selection queries with different selectivity factors, projections with different degrees of attribute duplication, 2-way and 3-way joins (including select/join queries and full joins), aggregates with and without grouping, and a handful of inserts, deletes, and updates. The database for the benchmark consisted of a set of synthetic relations with attribute value distributions that were designed to enable careful control over the selectivity-related and duplicate-value-related properties of the benchmark queries. The Wisconsin benchmark captured the attention of early relational database vendors, including INGRES, Britton-Lee, Oracle, and IBM, and it served as an important competitive forcing function that helped to drive industry progress in relational query optimization and execution in the early days of relational DBMS technology commercialization.

The Debit-Credit benchmark [32] was developed in the mid 1980's and was designed to test and compare the performance of DBMS transaction processing capabilities. In contrast with the synthetic nature of the Wisconsin benchmark, the Debit-Credit benchmark was a simple benchmark modeled after a banking application. The database for the benchmark consisted of account, teller, branch, and history files whose size ratios and data content were designed to scale in a fairly realistic manner based on the scale of the system being tested. The workload was multi-user, and it consisted of a number of teller terminals generating transactions at a fixed rate; the number of tellers was scaled up until the system being tested became unable to meet a specified goal for the response time distribution. The transaction rate (TPS) for that tipping point was reported as the system's transaction performance, and the cost of the system capable of providing that level of performance was also reported. Much as the Wisconsin benchmark did, the Debit-Credit benchmark captured the attention of the IT industry, and it served to drive significant industrial progress related to transaction processing performance. The Debit-Credit benchmark drew numerous industrial participants, including both software and hardware vendors, and it became so successful that it led to the formation of the Transaction Processing Council (TPC) in order to oversee the first two formal Debit-Credit inspired benchmarks, namely TPC-A and then TPC-B.

As DBMS technology and its functional richness have progressed over the past three decades, together with the performance of the underlying hardware and software platforms that these systems run on, a number of additional benchmarks have been developed with varying degrees of interest and adoption. On the strictly relational front, the TPC has produced a number of widely used benchmarks, including TPC-C, a more complex multi-user transaction processing benchmark based on an inventory management application, and TPC-H (formerly TPC-D), a single-user analysis query benchmark designed to test a system's complex query processing capabilities (somewhat in the spirit of the Wisconsin benchmark, being a series of queries, but with a database schema modeled after an enterprise data warehousing scenario). On the functionality front, a biased sub-sample of interesting benchmarks over the years might include the OO7 benchmark for object-oriented DBMSs [14, 15], the BUCKY benchmark for object-relational DBMSs [16], and the XMark and EXRT benchmarks for XML-related DBMS technologies [31, 17]. Each one of these benchmarks was a micro-benchmark based on an application-oriented database schema: OO7 was based on a computer-aided engineering design data management scenario, while BUCKY was based on a hypothetical university data management scenario; XMark considered an XML-based auction data management scenario, while EXRT based its choice of data on a financial services scenario that it borrowed from TPoX [28].

2.3 Existing BDMS Benchmarks

Owing to the importance of and interest in Big Data management solutions, work on benchmarking Big Data Management Systems (BDMSs) has started to appear. To date there have been two major benchmarking exercises that have caught the attention of a significant portion of the Big Data community, one in each of the two major sub-areas of Big Data – namely Big Data analytics and NoSQL data stores.

For Big Data analytics, the most influential study to date has been the work by Pavlo *et al* on "a Comparison of Approaches to Large-scale Data Analysis" [30] (which we will refer to henceforth as the CALDA effort, for brevity). This effort defined a micro-benchmark for Big Data analytics and then used it to compare the performance of Hadoop with that of two shared-nothing parallel DBMS platforms, a traditional parallel relational DBMS (denoted simply as DBMS-X in the study) and a parallel relational DBMS based on column-based data storage and column-oriented query processing techniques (Vertica). The data analysis tasks chosen for use in this benchmark were a *grep*-like selection task (borrowed from the original MapReduce paper), a range-based selection task, a grouped aggregation task, a two-way join task (including subsequent aggregation), and an aggregation task involving a user-defined function. The CALDA benchmark results included reporting of the times required to load and index the datasets for the benchmark, in the case of the two parallel database systems, as well as the execution times for each of the tasks on all three of the alternative Big Data analytics systems. A 100-node cluster was used in producing the reported initial benchmark results.

For NoSQL data stores, the most influential benchmark developed to date has been YCSB, the Yahoo! Cloud Serving Benchmark [20]. The goal of the YCSB effort was to create a standard benchmark to assist evaluators of NoSQL data stores, *i.e.,* of the wide range of new data stores that are targeting "data storage and management 'in the cloud'", based on scenarios of providing online read-write access to large volumes of simple data. YCSB is a multiuser benchmark that has two tiers, a performance tier and a scalability tier. YCSB's performance tier tests the latency of request processing for a loaded NoSQL system under workloads with mixes of reads (single-record gets and range scans) and writes (single-record inserts and updates). The system is tested as an open system; the rate of job arrivals is increased until the system becomes overloaded and response times are averaged per operation type in the workload's mix of operations. Several record popularity distributions are considered as well, including Uniform record access, Zipfian (by key), and Latest (Zipfian by insertion time, with recent records being more popular). The initial YCSB paper's main results came from running three workloads, an update-heavy workload (with 50%-50% reads and updates), a read-heavy workload (with 95% reads and 5% updates), and a short-range workload (with 95% range scans and 5% updates), against four different data stores: Cassandra, HBase, PNUTs, and MySQL. The scalability tier of YCSB examines the static and dynamic scalability of NoSQL systems. The static test is a scaleup test that varies the number of servers (from 1-12 in the initial study) while proportionally adding data as well. The dynamic test fixes the data size and the workload (which is sized to cause a heavy load when the system is small) and then increases the number of servers (from 2-6 in the initial study) over time in order to observe the performance of the system as more servers are added in order to absorb and balance the load on the system. Suggested future YCSB tiers included availability and replication testing.

In addition to these two benchmarks, other existing BDMS benchmarks include GridMix [3], a synthetic multiuser benchmark for Hadoop cluster testing, and PigMix [7], a collection of queries aimed at testing and tracking the performance of the Pig query processor from release to release. Also, the Web site [29] for the recent NSF Workshop on Big Data Benchmarking is a potentially useful resource for seekers of more information about existing Big Data benchmarks and/or about the community's thoughts on future needs and approaches in this area.

3 Lessons from Past Benchmarks (The Pitfalls)

The process of benchmarking Big Data systems, more traditional database systems, or most any computer software for that matter, is an interesting and challenging exercise.

Technical challenges often include somehow defining and agreeing upon acceptable domain- and or system-relevant notions of what is “reasonable”, “proper”, “fair”, “normal”, “comparable”, and/or “steady-state”. Challenges also include the level of detail at which a benchmark should be specified, or conversely, how much freedom should be left to eventual implementers of the benchmark. Non-technical challenges usually include dealing with others’ reactions to the benchmark, particularly from those individuals or organizations whose systems are being put to the test. This section of the paper discusses a number of potential pitfalls, mostly based on various of the author’s first-hand experiences over the years with benchmarking situations, that warrant consideration as this community strives to define useful and influential new Big Data benchmarks.

3.1 “Fair” Tuning Is Critical

One of the key challenges in conducting a benchmarking study, particularly one that aims to compare systems, is configuring all of the systems both “properly” and “fairly”. When the Wisconsin benchmark [22] was first being developed and run on a collection of early relational database systems, David DeWitt’s initial approach to configuring the systems was to run each one with its default configuration settings – *i.e.,* to base the benchmark numbers on each system’s “out of box experience”. The relational database systems being tested included the INGRES system, developed by Michael Stonebraker and Eugene Wong at UC Berkeley, the Britton-Lee IDM-500 database machine, developed by Bob Epstein and Paula Hawthorn, SQL DS, IBM’s commercialization of System-R, and Oracle, based on their own initial clone of the IBM System-R design. I was a graduate student at Berkeley and was in the “INGRES bullpen” on the day that Stonebraker got his first look at the initial DeWitt numbers, which were not very favorable for INGRES, and I seem to remember that it took several of us to pull him down from the ceiling after he’d looked at them. As it turned out, University INGRES was configured to run in a friendly (to other users) way on small Unix systems, so its buffer pool usage was modest and file-system based. In contrast, the IDM-500 was a dedicated box (with special search hardware as well as a very lightweight DB-oriented operating system), so it dedicated most of its vast main memory resources (something along the lines of 2MB ☺) to the buffer pool by default. In addition, the Wisconsin benchmark tables in those initial tests were very small, enough so that the numbers ended up comparing INGRES having to perform I/O against the IDM-500 running as a main-memory database system. I also later remember listening to a very angry Bruce Lindsay, from IBM’s System-R team, complaining passionately about how little sense it made to compare systems “out of the box” – as in those days, virtually no system was configured well “out of the box”. I didn’t hear Oracle’s reaction first-hand, but I do know that founder and CEO Larry Ellison apparently attempted to get David DeWitt fired from his faculty position at Wisconsin, so it seems he was not entirely pleased either (☺). The eventual published numbers from the Wisconsin benchmark study were produced using larger tables and only after setting the systems up as comparably as possible. Tuning is important! (It is also far from easy, as very often systems have different knobs, and it can be unclear how in fact to set them all across all systems to be “comparable”. We encountered challenges regarding memory settings, roughly thirty years after the initial Wisconsin benchmark, while trying to configure memory settings for several relational and non-relational database systems for the EXRT benchmark [17]).

3.2 Expect Unhappy Developers

Another challenge in conducting a benchmarking study is dealing with the human factor – which can easily escalate into the legal factor in some cases. When trying to address the first challenge, *i.e.,* ensuring proper use and tuning of each system, it can be very helpful to interact with experts from the companies or organizations whose software is going to be tested. Such individuals, who are often the lead developers of the products in question, are usually eager to make sure that things are done right, and usually start out being very helpful. Unfortunately, a benchmark is often viewed as a contest of some sort – in effect, it often is – and there must be winners and losers in any contest. Developers often become less enamored with a benchmark when it starts to turn up product "issues" that are going to be hard to address before the end of the study, and/or if their system's showing starts looking for any reason like it's not going to be the winner. This happened to David DeWitt, Jeff Naughton, and I when we worked together on the OO7 benchmark for object-oriented database systems [14, 15]. The systems that we tested included three commercial systems – each of us was actually on the Technical Advisory Board for one of those companies, so we had a "perfect storm" of conflicts of interest that cancelled one another out – so getting initial buy-in and cooperation was easy in each case. However, things got ugly later when the results started to emerge – to the point where Naughton took to shouting "Incoming!" whenever we heard the FAX machine across the hall get a call, as we began receiving threatening "cease and desist" orders from several of the companies' legal representatives. As it turns out, each had a "DeWitt clause" in their system's license agreement – a clause saying that one could not publically report performance results. We had not paid enough attention to this due to the cooperative attitudes of each company initially, but this clause gave the companies the power to order us to not publish results – and in fact, in the end, one of the companies did completely withdraw from the benchmark, and our university lawyers instructed us to not publish their results. Interestingly, the one company that opted out had been doing very well, the best in fact, performance-wise – its system just wasn't winning in absolutely every single test category. Since their product was the OODBMS market leader at the time, their management team decided that no good could come from participating in a contest they couldn't completely sweep – so the OO7 paper ended up having one less participant than it started out with. While this is less likely to happen today, at least for open-source Big Data systems (which don't have software licenses with "DeWitt clauses" in them), it is still almost certain that results from any given benchmarking effort will make the various systems' lead developers unhappy at some point.

3.3 Just How Declarative Is a Query?

Yet another challenge in developing a benchmark relates to defining and clearly specifying its operations. For SQL DBMS benchmarks, this is not a huge problem, as one can use SQL to specify the various operations in the benchmark. However, when one ventures into newer territories – like the Big Data territory of interest today – it's a different story. One might, for example, wish to come up with a "Big Data query benchmark" that can be run using "any" Big Data language – such as Pig, Hive, Jaql, or AQL – in which case this problem will arise. When DeWitt and Naughton and I were developing OO7, we faced this issue, as each OODBMS at that time had its own unique API as well as different query languages and capabilities. Some had fairly rich query languages, with expressive power comparable to SQL, while others had persistent, object-oriented programming language APIs with limited filtering options in their looping constructs (meaning that joins had to be hand-coded, among other implications, for the benchmark[1]). As a result, we ended up specifying the benchmark operations and our intentions in English [14, 15], as best we could, but this was not an entirely satisfying manner in which to specify a benchmark. Later, when we set out to benchmark object-relational systems in our Bucky benchmarking effort [16], we faced similar challenges – our work pre-dated the SQL3 standard, so there was no single query language, or even a truly uniform set of O-R extensions, to be used in our specification – so again we faced questions related to how to convey the properties "required" of a "correct" implementation of Bucky. Fast forward to the EXRT relational/XML benchmarking effort [17], of just a few years ago, and *still* this issue arose, albeit in a somewhat narrower form. In EXRT we tested several systems using only two standard XML query languages –SQL/XML and XQuery – so we were able to use those languages to write two specifications for each benchmark operation. However, what we found was that the systems, one of them in particular, were sensitive to the way that the queries were formulated – so we were faced with the question of whether or not it was "fair" to reformulate a query to work around a system's query optimizer blind spots and/or cost model glitches. The bottom line is that it's always something! This is sure to be a big issue for Big Data benchmarks, given the heterogeneity of current systems' languages and user models.

3.4 Is This a Reasonable Data Set?

When designing a set of operations for a benchmarking study, one needs data, so the design of a benchmark's database will essentially go hand-in-hand with the design of the benchmark itself. The Wisconsin benchmark used a completely synthetic set of tables (with table names like *1KTUP* and *10KTUP1* and column names like *unique1* and *tenpercent*) whose only purposes were to serve as targets for a series of carefully controlled queries. In contrast, as mentioned in Section 2.2, most of the DBMS benchmarks that followed have taken a more application-inspired approach to their

[1] We recently re-encountered this expressive power issue in an internal ASTERIX-related effort to compare our system's performance to that of MongoDB, one of the richest NoSQL stores [18]. The ASTERIX query language supports joins, but in MongoDB, joins have to be coded in client programs. As a result, we had some heated internal arguments about whether or not, and how well if so, to do that (*e.g.,* whether or not a typical client programmer would take the time to program a sort-merge or hash-based join method, and thus what a "fair" and/or "reasonable" implementation of a join would be for a user of MongoDB).

database designs, using a set of tables, a set of object collections, or a collection of documents intended to model something "real" drawn from a likely application area for the set of systems and features being tested. Typically these designs mix the realistic with the synthetic; attribute value distributions are still controlled to aid in the creation of benchmark queries with predictable performance and cardinality characteristics. An example of the latter approach is the XML database design used in the XMark [31] benchmarking effort, which is based on a hypothetical Web-based auction site scenario. As specified, the XMark database is a single and therefore potentially very large XML document containing nested collections of subdocuments about concepts such as people, world regions and items, item categories, and open and closed (finished) auctions. XMark's XML schema and data designs were based on careful consideration of various XML data features that transcend the relationally-representable norm and were therefore deemed to be interesting and important aspects of each system to test. So is this a reasonable design? At the time of its inception, it was felt that the answer was yes, but in retrospect, one could argue that having one humungous XML document containing an entire application database is probably both unrealistic and unwise from an application point of view. Big Data benchmarks will have to face similar choices and come to "reasonable" conclusions. In addition, since scale-up testing is an important aspect of Big Data testing, Big Data benchmarks are faced with the task of designing scale-up strategies for their data values – which is easily done for simple data, but can be quite challenging for data such as social graph data or data where fuzzy-matching properties (*e.g.,* entity-matching data sets) need to be maintained in a "realistic" manner as the data scale grows.

3.5 Steady as She Goes!

Well-engineered DBMSs in search of high performance employ techniques such as caching, or deferral and batching of certain operations, in their runtime systems. Database pages containing data are accessed from disk and then cached in memory (buffered) so that temporal locality can be leveraged to avoid subsequent I/Os for re-access; database queries are often compiled the first time they are encountered, and then their query execution plans are cached so that subsequent requests involving the same query, perhaps even with different input parameters, can avoid the cost of query planning by reusing the cached execution plan. For writes, most systems use defer-and-batch approaches at various levels in order to amortize write-related costs over multiple write operations. For example, transaction managers have long used group-commit techniques that delay individual transaction commits so as to commit multiple transactions with a single log write; the Vertica parallel DBMS and many of today's NoSQL data stores utilize LSM-based file structures (LSM = log-structured merge) so that write operations can first be performed on an in-memory tree component and the associated I/O costs occur later, asynchronously, in a batch, when the component is written to disk and/or merged with a disk-resident component of the file. The result of all of these optimizations is that the systems being benchmarked must be run for "long enough", with their initial warm-up period either being excluded or "drowned out", in order for the benchmark results to reflect the systems' steady-state behavior.

An example of how *not* to do this can be found in the last segment of the EXRT benchmark, which tests the performance of systems for a handful of simple update operations [17]. Two of the systems tested were traditional relational DBMSs with XML extensions, and they had traditional DBMS-like storage architectures and buffer managers; the third system was a native XML DBMS that has a different, LSM-like storage and caching architecture. The update-related performance results reported for the native XML system were much faster than for the relational systems due to this architectural difference and the fact that EXRT's update cost measurement approach allowed the update-related I/O's to "escape" beyond the measurement period. Being a micro-benchmark, each operation was run some modest number of times and then the results were averaged, and this simplistic methodology didn't properly capture the update I/O costs for the XML system's deferred-write architecture [17]. As we define new Big Data benchmarks, in a world with very large memory sizes and defer-and-batch mechanisms, benchmark designers need to be cognizant of the these techniques and their implications – and think about how to make benchmarking "fast enough" without losing track of important costs due to steady-state achievement issues.

3.6 Single- Versus Multi-user Performance

As we saw in Section 2's tour of prior benchmarks, some of the existing DBMS benchmarks have taken a single-user, query-at-a-time look at DBMS performance, *e.g.,* Wisconsin, OO7, Bucky, EXRT, while others have focused on the performance of DBMS's under multi-user workloads, *e.g.,* the TPC benchmarks A, B, and C. On the Big Data side, the CALDA evaluation of Big Data analytic technologies was a single-user micro-benchmark, while YCSB is a simple multi-user benchmark. When we started our ASTERIX project at UCI, one of our first steps was to make a set of "pilgrimages" with our initial project ideas to visit several major Big Data players – including one provider (Teradata) and several consumers (eBay and Facebook) – to get input on what considerations they viewed as important and what problems they thought we should be sure *not* to ignore in our work. One unanimous message that we received can be paraphrased as: "Real Big Data clusters are never run in single-user mode – they never run just one job at a time. Real clusters are shared and run a concurrent mix of jobs of different sizes with different levels of priority. Doing this sort of scheduling well is important, and nobody is truly there yet." This is important because decisions that one might make to optimize single-user, single-job response time in a Big Data system can be very different than the decisions that one would make once several instances of the job have to share the resources of the cluster, either amongst themselves or with other concurrent jobs. Several clear illustrations of the importance of multi-user thinking can be found in some performance work that we recently did related to a UCI Ph.D. student's summer internship at Facebook [25], where the student's assignment was to tweak Hadoop's runtime mechanisms so that analysts could run large exploratory HiveQL queries with `limit` clauses when investigating new questions and answers over large data sets in their daily work. The student's summer project led to Hadoop changes that enabled jobs to consume their input files in a more incremental fashion, and to cease their execution early when done.

These changes improved single-user Hive query performance on Hadoop only in cases where the job's needs outstripped the parallel resource capacity of the entire cluster (which of course can happen with Big Data's data sizes), but the changes *dramatically* improved Hive's multi-user performance in *all* cases, including multi-user cases with both homogeneous and heterogeneous job mixes sharing a cluster. We also looked briefly at how the default Hadoop scheduler's treatment of job mixes compared to that of one of the popular "fair schedulers" used by some Hadoop installations, and the results were surprising, at least to us – the multi-user performance of the "fair scheduler" was actually worse, at least for the workloads that we considered, due to its overly conservative utilization of the cluster's processing capacity.

3.7 Should the World Be Open and/or Classless?

Analytical and simulation-based modelers of the performance of computer systems have long faced questions about how to model a given system under study, and how to model the system's offered workload is an important question. One approach is to use an "open" system model – where jobs arrive and depart independently of how well the system is processing them. This is often an appropriate model when the workload for the system originates from a very large user base with very large per-user inter-request times, *e.g.,* as in a telephone network. Open systems can become overloaded when the load exceeds the system's capacity. Another approach is to use a "closed" system model – where there is a fixed population of users, each submitting one request, waiting for the system's response, thinking about the response, and then submitting their next request. This is often an appropriate model when the system has a more modest number of users, *e.g.,* where the system's user base is a small team of analysts each working interactively with a collection of data. A closed system can never become truly swamped in the fashion of an open system because its finite user population and the serial nature of requests mean that the system just "slows down" – users wait longer as the system's performance degrades, but they ultimately do wait for their answers before asking the next question. Another important workload modeling decision, for closed systems, is how to appropriately associate job classes with users – *i.e.,* whether each user can submit jobs of any class, or whether the system's active user population is better subdivided into different user classes, each with its own finite population, with each class of users submitting jobs with potentially different characteristics. The latter approach is arguably more appropriate for benchmarking DBMS performance under mixes of small and large jobs, *e.g.,* where small update requests or queries are mixed with large read-only requests. Otherwise, a scheduling policy that is somehow biased against one or the other of the workload's job classes may not have the opportunity to properly display its biases [13, 25], in which case the performance study may "miss" an important opportunity to surface a potential performance problem with one or more of the systems under study. This can occur because, without a static division of users into user classes, the system can eventually reach a state where every user is waiting for a response to a job of the class that the system is biased against – *i.e.,* those jobs can pile up, and eventually all users have an outstanding request of that type – and then it can finish at least one of those before it starts getting other kinds of requests again. Unless the actual load of the deployed system can be expected to have this "eventual relief" characteristic in production, it would not be good for a benchmark to give its studied systems such an "out" due to its choice of workload design.

4 Towards Future BDMS Benchmarks (The Possibilities)

Given the onslaught of Big Data, and the rapid expansion of Big Data as a platform sector in the IT landscape, opportunities abound for future efforts related to Big Data benchmarking [29]. Given the current status of the work in this area, and in light of the potential pitfalls just discussed, we close this paper by briefly surveying some of the unmet needs and important future possibilities (at least in this author's opinion) regarding BDMS performance characterization and assessment efforts.

Richer BDMS Micro-Benchmarks: The CALDA effort on benchmarking Big Data analytics technologies was a good start, clearly, but it really just scratched the surface in this area. There is a clear need for a more comprehensive benchmark, perhaps along the lines of the Wisconsin benchmark or OO7, that could be used in evaluating the emerging generation of BDMS alternatives. Such an effort would consider a much broader range of queries and updates, including small- and medium-sized requests against indexed data in addition to large, MapReduce-influenced jobs.

Multi-user BDMS Benchmarks: The YCSB effort on benchmarking Big Data key-value stores was a good start as well, but again it represents a surface scratch relative to the ongoing needs in this area; YCSB studied only a few workload mixes involving very simple operations. Needs here include profiling the workloads of operational clusters in order to identify the kinds of query mixes seen in Big Data analytical settings, in practice, on shared clusters. The results of such profiling could then be used to create synthetic workloads for evaluating the existing Big Data technologies in realistic settings as well to help drive research on cluster resource management. Other needs include the exploration of multi-user workloads involving mixes of both analytical and simple data access requests, *i.e.,* truly heterogeneous workloads.

Domain-Centric Big Data Benchmarks: Social graph data, scientific data, spatial data, streaming data (*a.k.a,* "Fast Data" or "high-velocity data") – each of these is the current focus of one or more specialized Big Data platform efforts. To help assess the platform progress in these areas, and to help motivate their engineering efforts, new benchmarks are needed. In each case, the new programming models being proposed, and the associated new data handling mechanisms being developed for them, are sufficiently specialized to justify the development of specialized benchmarks as well. (The "Linear Road" benchmark [2] from the stream data management community is one example of what such a specialized benchmark might look like.)

Self-Management Benchmarks: Some of the most attractive features of scalable distributed file systems like HDFS and of key-value stores like HBase and Cassandra are their support for auto-management of storage as new data and/or cluster nodes are added and their support for high availability in the face of data node failures. Good benchmarks that can be used to compare and evaluate these dimensions of Big Data platform functionality are an open problem, and will be important to drive research and development efforts in these areas.

Challenging Data Benchmarks: Last but not least, the emerging new generation of BDMS platforms is starting to provide very rich functionality in areas such as flexible schema support, fuzzy searching and matching, and spatial data handling. Again, the result is an opportunity to develop new benchmarks, benchmarks that can proxy for the requirements of emerging Big Data applications, in order to evaluate and drive the work being done in these key new areas of BDMS functionality.

Acknowledgments. The work to date on the ASTERIX project at UC Irvine has been supported by NSF IIS awards 0910989 and 0844574, by a grant from the University of California Discovery program with a matching donation from eBay, by a Facebook Fellowship award and several graduate Yahoo! Key Scientific Challenges awards, and through the provision of access by Yahoo! Research to one of their large research computing clusters. The ideas shared here owe much to discussions over the years with the co-developers of prior benchmarks, particularly David DeWitt and Jeffrey Naughton, as well as with the co-leads of the ASTERIX project, particularly Vinayak Borkar and Chen Li at UCI.

References

1. Alsubaiee, S., Behm, A., Grover, R., Vernica, R., Borkar, V., Carey, M., Li, C.: ASTERIX: Scalable Warehouse-Style Web Data Integration. In: Proc. Int'l. Workshop on Information Integration on the Web (IIWeb), Phoenix, AZ (May 2012)
2. Arasu, A., Cherniack, M., Galvez, E., Maier, D., Maskey, A., Ryvkina, E., Stonebraker, M., Tibbetts, R.: Linear Road: A Stream Data Management Benchmark. In: Proc. VLDB Conf., Toronto, Canada (August 2004)
3. Apache GridMix, `http://hadoop.apache.org/mapreduce/docs/current/gridmix.html`
4. Apache Hadoop, `http://hadoop.apache.org/.`
5. Apache Hive, `https://cwiki.apache.org/confluence/display/Hive/Home`
6. Apache Pig, `http://pig.apache.org/.`
7. Apache PigMix, `https://cwiki.apache.org/confluence/display/PIG/PigMix`
8. ASTERIX Project, `http://asterix.ics.uci.edu/.`
9. Behm, A., Borkar, V., Carey, M., Grover, R., Li, C., Onose, N., Vernica, R., Deutsch, A., Papakonstantinou, Y., Tsotras, V.: ASTERIX: Towards a Scalable, Semistructured Data Platform for Evolving-World Models. Distrib. Parallel Databases 29(3) (June 2011)
10. Borkar, V., Carey, M., Grover, R., Onose, N., Vernica, R.: Hyracks: A Flexible and Extensible Foundation for Data-Intensive Computing. In: Proc. IEEE ICDE Conf., Hanover, Germany (April 2011)
11. Borkar, V., Carey, M., Li, C.: Inside "Big Data Management": Ogres, Onions, or Parfaits? In: Proc. EDBT Conf., Berlin, Germany (March 2012)

12. Bu, Y., Borkar, V., Carey, M., Rosen, J., Polyzotis, N., Condie, T., Weimer, M., Ramakrishnan, R.: Scaling Datalog for Machine Learning on Big Data. arXiv:1203.0160v2 (cs.DB) (March 2012)
13. Carey, M., Muhanna, W.: The Performance of Multiversion Concurrency Control Algorithms. ACM Trans. on Comp. Sys. 4(4) (November 1986)
14. Carey, M., DeWitt, D., Naughton, J.: The OO7 Benchmark. In: Proc. ACM SIGMOD Conf., Washington, DC (May 1993)
15. Carey, M., DeWitt, D., Kant, C., Naughton, J.: A Status Report on the OO7 OODBMS Benchmarking Effort. In: Proc. ACM OOPSLA Conf., Portland, OR (October 1994)
16. Carey, M., DeWitt, D., Naughton, J., Asgarian, M., Brown, P., Gehrke, J., Shah, D.: The BUCKY Object-Relational Benchmark. In: Proc. ACM SIGMOD Conf., Tucson, AZ (May 1997)
17. Carey, M.J., Ling, L., Nicola, M., Shao, L.: EXRT: Towards a Simple Benchmark for XML Readiness Testing. In: Nambiar, R., Poess, M. (eds.) TPCTC 2010. LNCS, vol. 6417, pp. 93–109. Springer, Heidelberg (2011)
18. Cattell, R.: Scalable SQL and NoSQL Data Stores. ACM SIGMOD Rec. 39(4) (December 2010)
19. Chaiken, R., Jenkins, B., Larson, P., Ramsey, B., Shakib, D., Weaver, S., Zhou, J.: SCOPE: Easy and Efficient Parallel Processing of Massive Data Sets. Proc. VLDB Endow. 1(2) (August 2008)
20. Cooper, B., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R.: Benchmarking Cloud Serving Systems with YCSB. In: Proc. ACM Symp. on Cloud Computing, Indianapolis, IN (May 2010)
21. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. In: Proc. OSDI Conf. (December 2004)
22. DeWitt, D.: The Wisconsin Benchmark: Past, Present, and Future. In: [24]
23. DeWitt, D., Gray, J.: Parallel Database Systems: The Future of High Performance Database Systems. Comm. ACM 35(6) (June 1992)
24. Gray, J.: Benchmark Handbook for Database and Transaction Systems, 2nd edn. Morgan Kaufmann Publishers, San Francisco (1993)
25. Grover, R., Carey, M.: Extending Map-Reduce for Efficient Predicate-Based Sampling. In: Proc. IEEE ICDE Conf., Washington, D.C (April 2012)
26. Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C., Hellerstein, J.: GraphLab: A New Parallel Framework for Machine Learning. In: Proc. Conf. on Uncertainty in Artificial Intelligence (UAI), Catalina Island, CA (July 2010)
27. Malewicz, G., Austern, M., Bik, A., Dehnert, J., Horn, I., Leiser, N., Czajkowski, G.: Pregel: A System for Large-Scale Graph Processing. In: Proc. ACM SIGMOD Conf., Indianapolis, IN (May 2010)
28. Nicola, M., Kogan, I., Schiefer, B.: An XML Transaction Processing Benchmark. In: Proc. ACM SIGMOD Conf., Beijing, China (June 2007)
29. NSF Workshop on Big Data Benchmarking, `http://clds.ucsd.edu/wbdb2012/`
30. Pavlo, A., Paulson, E., Rasin, A., Abadi, D., DeWitt, D., Madden, S., Stonebraker, M.: A Comparison of Approaches to Large-Scale Data Analysis. In: Proc. ACM SIGMOD Conf., Providence, RI (June 2009)
31. Schmidt, A., Waas, F., Kersten, M., Carey, M., Manolescu, I., Busse, R.: XMark: A Benchmark for XML Data Management. In: Proc. VLDB Conf., Hong Kong, China (August 2002)
32. Serlin, O.: The History of DebitCredit and the TPC. In: [24]
33. Stonebraker, M., Brown, P., Poliakov, A., Raman, S.: The Architecture of SciDB. In: Proc. SSDBM Conf., Portland, OR (July 2011)