TPCx-BB (Big Bench) in a Single-Node Environment

Dippy Aggarwal[1(✉)], Shreyas Shekhar[1], Chris Elford[1], Umachandar Jayachandran[2], Sadashivan Krishnamurthy[2], Jamie Reding[2], and Brendan Niebruegge[2]

[1] Intel Corporation, Santa Clara, OR, USA
{dippy.aggarwal,shreyas.shekhar,chris.l.elford}@intel.com
[2] Microsoft Corporation, Redmond, USA
{umachandar.jayachandran,skrish,jamie.reding,brnieb}@microsoft.com

Abstract. Big data tends to concentrate on the data volume and variety which requires large cluster capabilities to process diverse and heterogeneous data. Currently, NoSQL/Hadoop-based cluster frameworks are known to excel at handling this form of data by scaling across nodes and distributed query processing. But for certain data sizes, relational databases can also support these workloads. In this paper, we support this claim over a popular relational database engine, Microsoft* SQL Server* 2019 (pre-release candidate) using a big data benchmark, BigBench. Our work in this paper is the industry first case study that runs BigBench on a single node environment powered by Intel® Xeon™ processor 8164 product family and enterprise-class Intel® SSDs. We make the following two contributions: (1) present response times of all 30 BigBench queries when run sequentially to showcase the advanced analytics and machine learning capabilities integrated within SQL Server 2019, and (2) present results from data scalability experiments over two scale factors (1 TB, 3 TB) to understand the impact of increase in data size on query runtimes. We further characterize a subset of queries to understand their resource consumption requirements (CPU/IO/memory) on a single node system. We will conclude by correlating our initial engineering study to similar research studies on cluster-based configurations providing a further hint to the potential of relational databases to run reasonably scaled big-data workloads.

Keywords: TPCx-BB · Microsoft* SQL Server* 2019 · Big data · BigBench · Machine learning · Natural language processing

1 Introduction

With the popularity of Big Data, organizations are continuously confronted with the challenge of storing, processing and analyzing diverse, complex form of data (structured, semi-structured, and un-structured) with relatively limited storage and computation power of traditional tools and processes when compared

R. Nambiar and M. Poess (Eds.): TPCTC 2019, LNCS 12257, pp. 64–83, 2020.
https://doi.org/10.1007/978-3-030-55024-0_5

to cluster-based configurations. There are currently two alternatives to handle analysis over this form of data: (a) scale-up (adding CPU power, memory to a single machine), (b) scale-out (adding more machines in the system creating a cluster). Several commercial and open-source systems have emerged over the past several years as a scale-out solution to handle this data form [25–30]. Additionally, researchers from different communities including data management, machine learning, systems and computer architecture are also continuously working towards proposing innovative approaches to manage big-data workloads [31–35].

On one hand, having a variety of solutions is an advantage, for the flexibility they offer big data users in making a choice for the solution that best fits their needs; on the other hand, it also presents the challenge of assessing the functionality and performance of different systems and having a comparative view to even arrive at a decision. Benchmarking is a standard process embraced by organizations and researchers for measuring and demonstrating performance of their products. Additionally, it helps to identify the potential areas for performance optimizations, drive engineering efforts, and through this process, provide insights into the requirements for the next version of product/system under test.

Several big data benchmarks have been proposed recently out of which BigBench is considered as the first end-to-end solution designed to evaluate entire systems [5,6,10]. There exists another category of benchmarks called microbenchmarks. They are designed to evaluate specific system components or features of Big Data solutions For example, NNBench [8] tests the performance of NameNode component in HDFS, Terasort [9] is another benchmark used to evaluate performance of MapReduce framework by measuring the amount of time it takes to sort one terabyte of randomly distributed data. Similarly, there are several other active efforts in the benchmarking community that are mainly targeted at evaluating specific components [5,6].

Han et al. [6] present a survey of open-source big data benchmarks. The authors classify the surveyed benchmarks across four different dimensions: (1) types of big data systems they serve (Hadoop-based, DMBSs/NOSQL data stores, and specialized systems which require processing on particular data types such as graphs, streaming data), (2) workload generation techniques, (3) input data generation techniques, and (4) performance metrics. In another literature study, Ivanov et al. [5] provide a summary of big data benchmarks discussing their characteristics and pros and cons. Ghazal et al. [7] highlight three specific areas that make BigBench unique when compared to the other big data benchmarks. Along with those three attributes (technology-agnostic, end-to-end benchmark as opposed to a microbenchmark, coverage of the workload to address variety aspect of big data), one other aspect that distinguishes BigBench from other big data benchmarks is that it has been endorsed and adopted by a standardized benchmark committee, TPC* (Transaction Processing Performance Council).

*Other names and brands may be claimed as property of others

In the database world, TPC is the organization that is responsible for defining, publishing, auditing, and maintaining database benchmarks. TPCx-BB* was originally created in 2016 to showcase Hadoop style big-data analytics cluster Scale Out scenarios. It is based on a scientific proposal, BigBench, which was later adopted by TPC as an industry-standard benchmark [42]. The mostly read query mix and data consistency guidelines make it very friendly to scale-out Hadoop style implementations. It is unique compared to other influential database benchmarks defined by TPC in two ways. First, the other benchmarks such as TPC-H*, TPC-DS* are pure SQL benchmarks while TPCx-BB includes additional processing paradigms (machine learning and natural language processing) and data formats (unstructured and semi-structured) which emulates the text of user product reviews and web clickstream logs for an e-commerce site. BigBench uses this data for a number of machine learning and natural language processing queries to mine the reviews for interesting artifacts, data classification and clustering. Second, TPCxBB is an express benchmark which means that it not only comes with a specification document listing functional requirements, but also offers a ready to use executable kit that can be quickly deployed and used for benchmarking [24]. The original implementation of BigBench uses Hadoop and Hive [15].

Since its adoption by TPC, several organizations have published results of their studies using this benchmark to demonstrate competitiveness of their products in the big data market [10–13,19]. However, all of these contributions address a cluster-based configuration with data processing split across multiple nodes. The original TPCx-BB implementation provided on the TPC site also uses Hadoop-based framework and processing engine, Hive that requires a cluster setup. Rabl et al. [43] provide a proof of concept of running 30 BigBench queries on a Teradata Aster Database which executes queries using SQL-MR (Map Reduce) interface. The tests were conducted on a 8 node cluster.

Certain database use cases such as those emulated by TPCx-BB/BigBench with limited data sharing dependencies are amenable to scale-out and achieve performance benefits by parallelizing over a cluster of [typically cheaper] servers working together to process “large” to “enormous” databases. This is in contrast to traditional OLAP benchmarks like TPC-H which feature more data sharing dependencies and showcase the scale up potential of a single [typically more expensive] server being able efficiently process “medium” to “large” databases. The definition of “medium”, “large”, and “enormous” datasets is quite subjective and changes over time leaving a fair amount of overlap between use cases designed for scale-out via Hadoop type design and scale-up via traditional database design. With the advancements in hardware technologies and how the database performance is heavily influenced by the underlying resources available to it, we investigate the potential of running BigBench on a single node platform as opposed to a cluster-based configuration.

We present our preliminary results from running BigBench on a system powered by Intel technologies and a relational database engine, Microsoft* SQL Server* 2019 (pre-release). The goal is to demonstrate advanced analytics capa-

bilities and performance of SQL Server* 2019 and reinforce the position that scale-up remains an option for an important subset of database sizes. We make the following two contributions:

1. We present the results of running all 30 BigBench queries over a relational database engine and a single node configuration powered by Intel technologies. It serves to illustrate the functional capabilities of Microsoft* SQL Server* engine for processing queries that go beyond pure relational, SQL paradigm.
2. We present our preliminary results from experiments designed to understand the impact of increase in input dataset size on query runtime. We analyze the results across 1 TB and 3 TB. We further analyze the runtime behavior of a subset of queries over 3 TB to understand their resource consumption requirements (CPU/IO/memory) on a single node system.

It is important to note that we do not reference any official TPCx-BB benchmark metrics and our results only include query execution times since official metrics reporting is only permitted in TPC-audited results.

The remainder of this paper is organized as follows. Section 2 covers the necessary background concepts including Microsoft* SQL Server* 2019 advanced features that make it amenable to big data workloads such as BigBench and an overview of BigBench data model and queries. We discuss our single node based experimental setup in Sect. 3 with our results in Sect. 4. Section 5 presents the work related to big data benchmarking and compares our initial results with existing academic studies that have all been over cluster-based environment. Finally, Sect. 6 summarizes the paper with ideas for future work.

2 Background Concepts

In this section, we first present concepts that are important to understand the terms referenced later in the paper. These include (1) Microsoft SQL Server 2019 Extensibility Framework which enables execution of machine learning and natural language processing algorithms from within SQL Server, and (2) an overview of TPCxBB data model and queries.

Microsoft SQL Server Extensibility Framework. Using machine learning libraries to build prediction and forecasting models and perform statistical operations is a common scenario these days and R and Python are the two most popular languages used for machine learning. However, there are two inherent challenges while processing data using R/Python libraries: (1) complete data needs to be loaded in memory before any computation can be performed, (2) data needs to be available/moved to the server where R/Python runtimes are installed. Microsoft alleviates these issues for the data resident in SQL Server by introducing external, machine learning execution engines (R/Python) and libraries as a part of SQL Server offering. This feature is called machine learning extensibility framework and gives user the ability to execute code in an external runtime engine without leaving SQL Server environment.

Fig. 1. Architecture Diagram for Microsoft* SQL Server* Third Party (Java) Extensibility Framework [38]

Starting with SQL Server* 2016, Microsoft introduced extensibility framework as an add-on to the core database engine thus allowing execution of machine learning algorithms implemented in R from within SQL Server*. Microsoft's implementations of machine learning algorithms (RevoScaleR library) allows processing of datasets that may not completely fit in memory [39]. BigBench includes five machine learning queries (5, 20, 25, 26, 28) covering clustering, logistic regression, and classification. SQL Server* 2019 added support for yet another external runtime engine, Java, to the extensibility framework that now allows the user to run pre-compiled Java code fully integrated within core database query execution [40]. BigBench workload includes four queries (10, 18, 19, 27) which require the use of open-source java libraries. Having built-in support for java extensions and machine learning algorithms within SQL Server allows efficient execution and use of those libraries for benchmarking SQL Server database system using the bigbench workload.

Figure 1 shows the architecture for extensibility framework (specifically Java). Machine learning algorithms in R or Python also uses similar framework with a few differences explained as follows. There are essentially three major components that are involved in execution of R, Python, or Java code from within SQL Server. These include Launchpad, external runtime process, and SqlSatellite. The data exchange between SQL Server and external runtime is managed by SqlSatellite. A call to an external runtime process from SQL Server is initiated using a built-in stored procedure sp_execute_external_script provided by SQL Server. It initiates Launchpad which is a service that comes in packaged within SQL Server when we choose to include Machine Learning services with it. The launchpad service further starts a launcher dll which is either RLauncher.dll, PythonLauncher.dll, commonlauncher.dll depending on the external runtime user wishes to invoke. Microsoft allows only trusted launchers which are either published or certified by Microsoft. This constraint serves two purposes. First, the trusted launchers are guaranteed to meet performance and resource management requirements established by SQL Server. Second, the policy ensures security of the data residing in SQL Server. Although the external runtime executes in a separate process from SQL Server, but it is a part

of the SQL server framework which facilitates data access and operations on the server. In case of external code based on Java, commonlauncher.dll initiates and manages a process ExtHost which performs two functions: (1) uses sqlsatellite for communication and providing data/scripts to the extension code from SQL Server, (2) hosts another module, Language Extension which loads and communicates with JVM/Java-based user code.

A query in SQL Server that uses this extensibility framework and parallel execution mode may result in a large number of concurrent external processes being provisioned to facilitate parallelism and satisfy the query plan. We present two BigBench queries in appendix to highlight SQL Server* capabilities around both machine learning and natural language processing queries.

BigBench Workload. BigBench is the scientific research proposal which was later adopted by TPC to define an end-to-end, express benchmark, TPCx-BB. It is end-to-end because it is designed to evaluate performance of complete system as opposed to specific components or features of system under test. The benchmark models a fictitious retailer company selling products to customers via both physical and online stores. Twenty queries out of thirty in the workload address big data uses cases defined in McKinsey report [16] and the remaining ten queries are based on another decision-support benchmark defined by TPC, known as TPC-DS [10,14].

In our experiments, we demonstrate results of 30 BigBench queries when run sequentially with a single query running at any given point in time. This portion of the benchmark is called "power run". We capture data across two scale factors: 1000 and 3000 (1 TB and 3 TB). We collect each query's start, end, and execution times, and data on the resource utilization for the complete run of thirty queries using Microsoft Windows Performance Tool, perfmon [17]. These two datasets (perfmon and query runtime information) when analyzed together allows us to identify performance of each query over the complete duration of the run.

3 Experimental Setup

In this section, we describe the system setup that we used for running our experiments.

Hardware: We use a single node system running Windows Server 2016 Datacenter with a pre-release candidate of Microsoft SQL Server 2019 (CTP 2.4) installed as the database engine. The system is equipped with a 4-socket Intel® Xeon™ Platinum 8164 processor (2 GHz), each with 26 physical cores (total 104 cores), 3 TB main memory (48 DIMMs of 64 GB each, Frequency: 2400 MHz), and up to 40 TB of storage powered by Intel® SSDs. We believe that the current setup can be further tuned and there is potential for additional performance gains even with the current experiments. This paper focuses on presenting our initial results with the goal of showcasing the potential of scale-up configurations for big data workloads.

Our storage configuration is as follows. We have two storage bays with each bay populated with 24 Intel® SSD drives. The configuration is shown in Table 1. For each bay, we created RAID0 across 7 × 800 GB Intel® SSD drives for storing database files (10 TB total storage for database data files) and followed same configuration for tempdb drives (10 TB total storage for tempdb files). TempDB is configured and placed on separate drives from the database files (mdf) to isolate any the impact of any spills that may occur during query execution and RAID0 ensures good performance. At query runtime, data is read in parallel from the two data drives. Logical drives for storing raw data (flat files)were created using 6 × 800 GB Intel® SSD drives resulting in 4.36 TB storage on each storage bay. The remaining 4 SSDs of 1.5TB each were used to storage backup files. Lastly, for SQL log files, we included a single 800 GB Intel® SSD.

Table 1. Storage configuration details per storage bay

File type	SSD details	# of files per file type
Datafiles/Tempdb files/Flat files/Log	Intel® SSD DC S3700 (800 GB)	7/7/6/1
Backup files	Intel® SSD DC S3500 (1.6 TB)	4

Software. We enabled following two features for SQL Server: *lock pages in memory* and *large pages* (-T834 flag). Using large pages allows SQL Server to allocate allowed memory capacity at the time of startup and leverage CPU support for large virtual memory pages. The amount of memory allocated by SQL Server is equal to the minimum of 'max server memory' and physical memory available on the system. We set the max server memory to 2.4 TB for our experiments leaving $\tilde{6}$00 GB for other processes running on the system and execution of machine learning services from within SQL Server [41]. This practice improves query runtime since any memory required by a SQL query at runtime is not dynamically allocated from OS and initialized at the time of execution, but is readily available for use.

In terms of the available memory for query execution, we configured memory grant % in resource governor settings in SQL Server, setting it to 75% [22]. This ensures that each individual query can get maximum of 75% of the available memory to SQL Server instance.

4 Results

In this section, we present our results from running TPCxBB workload over two different scale factors: 1000, 3000. The experiments were conducted using the schema and query implementations that were specifically developed to run BigBench on SQL Server. Since the original implementation of BigBench supports Hive on MapReduce, queries had to be translated to enable running them on a relational engine, SQL Server in our case. We first present our data scalability experiments to understand the trends in query response times with increase in

dataset size. Next, we discuss hardware resource utilization for a subset of queries which stresses different categories of resources. The methodology we followed for collecting data for each scale factor is a restart of SQL server in order to clear the buffer pool, followed by running all 30 queries sequentially in ascending order. Table 2 shows contribution of each query group (Pure SQL, ML, NLP) to the total runtime for 30 queries.

In terms of the total elapsed time for the complete run, 1 TB took 11,712 s ($\tilde{1}95$ min) while it took 44,376 s($\tilde{7}39$ min) with a 3 TB scale factor. Baru et al. [10] present results from running BigBench on a 6-node Cloudera cluster (20 cores and 128 GB main memory per node) with Hive/MapReduce framework. The 30 queries run in sequential order on their configuration took more than 1200 min. They also conducted similar study on a larger cluster with 544 nodes (12 cores and 48 GB main memory per node) and it took more than 200 min to run all 30 queries. Our experiments were run on a single node with lesser number of cores compared to each of these cluster setups, yet finishes in less time. One of the reasons for this performance gain can be attributed to the foundational schema-based nature of relational databases. In Hadoop based systems, data is directly loaded onto HDFS in raw format without assigning any particular structure to it. While this often results in faster data load times, eventually the cost manifests at the query runtime since the data needs to be parsed to be able to pull only the required information.

Table 2. Query runtimes % per query group vs. scale factor (ML and NLP queries scale better than SQL)

Group	1TB	3TB
NLP	63.89	41.90
ML	6.16	3.08
SQL	29.95	55.02

In another study conducted in Frankfurt Big Data Lab [18], researchers used BigBench on a 4 node cluster with a total of 30 cores/60 threads and 128 GB total memory and using Hive/MapReduce as the execution framework. Absolute runtimes of all 30 queries were reported showing how a single query (query 4) by itself took more than 900 min. While our results can not be directly compared to the 4-node cluster setup because of the difference in hardware configurations, it still points to the potential of scale up configurations to run reasonably sized big data workloads.

The other important point to note in Table 2 is how SQL queries dominate the overall query runtime with increase in scale factor. We elaborate on this behavior in the discussion around scalability experiments for specific query groups.

Data Scalability for ML Queries. While Table 2 presents a high-level picture, next we focus on specific queries and organize our results per query group. Figure 2 shows the scaling behavior of all 30 queries over scale factors: 1000 and 3000. In this section, we focus on five machine learning queries (5, 20, 25, 26, 28). These queries cover clustering (20, 25, 26), regression and classification (5, 28). All the five queries scale well (up to 2x) with overall 3x increase in data size. While additional data from higher scale factors such as 10 and

30 TB are required to validate the trend further, the current performance we are observing can be attributed to following two areas within SQL Server: (1) the distributed and parallel computation of machine learning algorithms based on Microsoft implementations of these algorithms (RevoScaleR library) and the data exchange model used between SQL Server and Machine Learning engine. (2) multi-threaded execution of SQL queries which read and join data from the database tables to prepare input data for ML algorithms used later in the query.

Data Scalability for NLP Queries. In this section, we focus on the scaling behavior of natural language processing queries (10, 18, 19, 27) over scale factors: 1000 and 3000 in Fig. 2. Except query 18, rest three queries all scale linearly or even better with increase in scale factor. This seems to be an artifact of the amount of data that is actually passed to the java extensions framework for running the underlying pre-compiled Java code for negative sentiment analysis required in this query. Query 18 uses a set of SQL operators including joins, scans, and filters to prepare the data for sentiment analysis using Java and that input data shows 6× increase (4,013,405 vs. 24,634,495 rows) between scale factors 1000 and 3000. In this context, the query did scale well ($\tilde{4}\times$ for $\tilde{6}\times$ increase in data size). In our current set of experiments for NLP queries, we restricted parallelism and used lesser number of cores than available on the system and we plan to investigate this further to seek further optimizations.

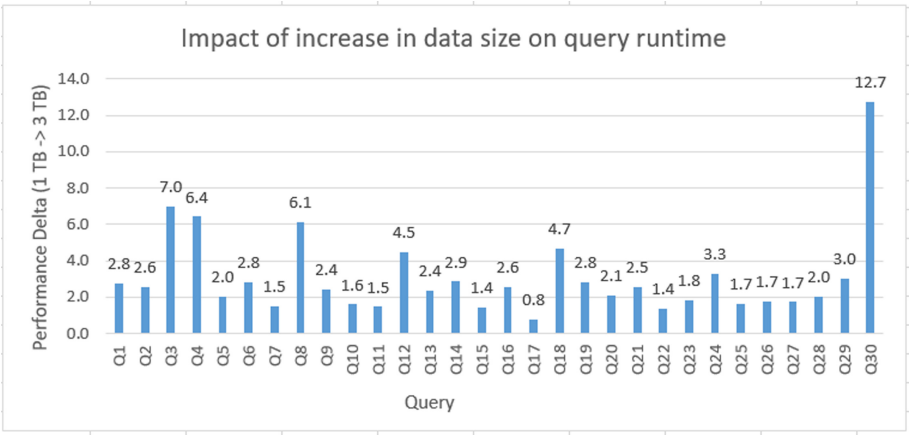

Fig. 2. Scalability of BigBench queries with increase in scale factor

Data Scalability for Pure SQL Queries. Having discussed NLP and ML queries, we now discuss scalability trends for SQL queries. Queries 3, 4, 8, 12, and 30 appear as the worst scaling queries among the 21 SQL queries. For query 8, even though the runtime increased by 6× from 1 TB to 3 TB, the absolute

runtimes are relatively small (11.89 s for 1 TB and 77.87 s for 3 TB). Similarly query 12 takes 4× time to finish with 3x increase in data size. But again, the absolute run times are quite short (4.95/22.27 s). We will elaborate on the remaining three queries (3, 4, 30) and query 2 which seems to be scaling well. Figure 5 offers an insight into query characteristics for 3, 4 and 30.

Query 3 shows a 7× increase in runtime with only 3x increase in data size. This query reads data from one of the largest tables, web_clickstreams. On analyzing resource utilization pattern from performance monitor, the query consistently shows high CPU utilization (95%) over both scale factors with minimal disk activity. This points to the fact that running it on a system with even higher number of cores would potentially yield performance gains unless it is busy waiting, i.e. spinning. We plan to investigate this behavior further.

Query 4 is memory bound with spills to tempdb at both 1 TB and 3 TB scale factors. The memory grant information from query plans also shows the memory pressure observed during this query. The maximum available memory for any query based on our current settings is 1.3 TB where as query 4 desires 7 TB memory for 1 TB scale factor and it needs 22 TB for 3 TB. The CPU utilization for this query decreased from 37% on average on 1 TB to 21% on 3 TB, and Figures 3 and 4 show the reason behind this drop in utilization. It is either during the time period where query was writing to Tempdb or during a phase where there is no Tempdb activity but utilization is still low. This is coming from few single-threaded operations including stream aggregate. The high memory capacity requirement highlighted here captures the characteristic of the query as opposed to bottlenecks in underlying hardware or SQL Server processing model.

Query 30 shows similar resource utilization behavior as query 4. There is a drop in CPU utilization because of the increased I/O activity with Tempdb at 3 TB.

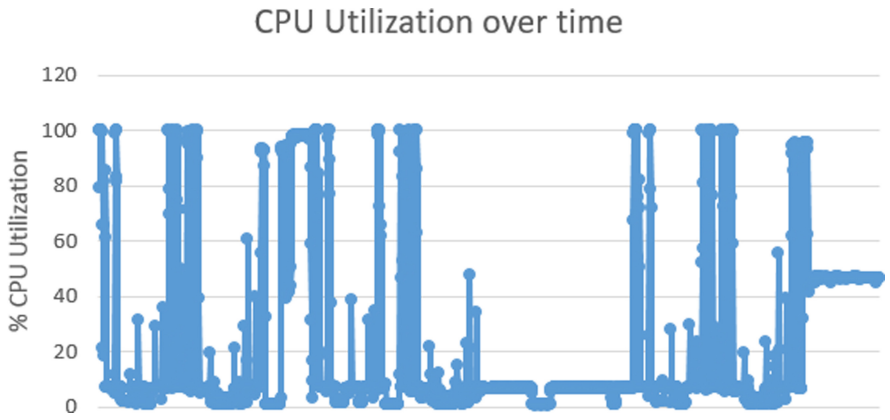

Fig. 3. CPU Utilization for query 4 on 3 TB scale factor (Low average utilization: 21%)

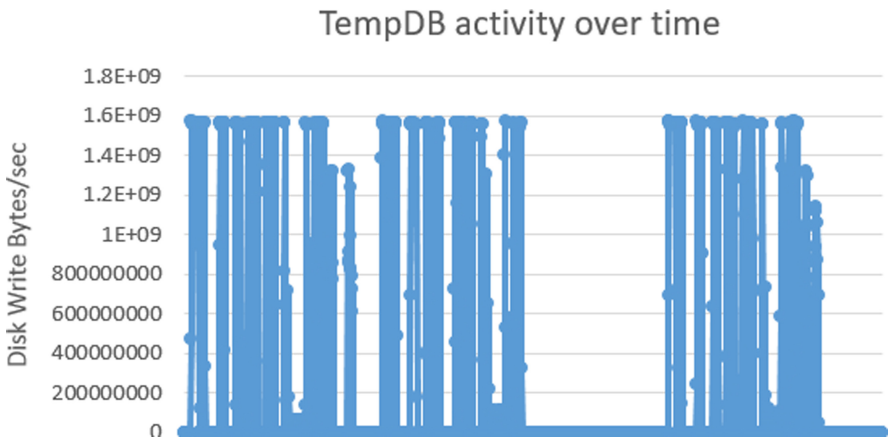

Fig. 4. TempDB write activity over 3 TB for query 4 - Disk Write Bytes/sec (overlaps with few areas of low CPU utilization from Figure 3)

While queries 3, 4, and 30 offer scope for further tuning for performance optimizations, there are also queries which showed positive scaling behavior (queries 2, 11, 15, 22). Queries 11, 15, 22 do not show much variation in their run times with the increase in data size. The runtime for query 2 is increased by 2.5× (101 vs. 259 s) while the input datasize went up three times. Analyzing the resource consumption for this query, we observe that the CPU utilization increased from 21% on average on 1 TB to 34% on average on 3 TB, highlighting the performance benefit gained from a system with high number of cores. However, there is also disk activity coming from spills to Tempdb (16 GB on 1 TB vs. 82 GB on 3 TB) that is impacting query runtime. But overall, query 2 does scale well.

This concludes our discussion on query behavior and performance over two different scale factors. While most of the queries scale well, we do identify few queries (3, 4, 30) that may have additional opportunities for optimization. In addition, running experiments over higher scale factor such as 10 TB might help in highlighting those areas as well.

5 Related Work

Given that BigBench is designed for evaluating big data systems, our survey of existing contributions is directed along the following dimensions: (1) identify related studies conducted on big data systems and which benchmarks did they employ for benchmarking, (2) identify studies focused on BigBench, and (3) performance studies focused on single node SQL Server and the benchmarks that were employed for those studies.

For the first case, there exists several contributions that focus on evaluating and comparing various SQL-on-Hadoop systems and many of them use either TPC-H, TPC-DS inspired benchmarks [1–3,36], or microbenchmarks as

			DataDB		TempDB	
Query		CPU Utilization	Data read (GB)	Data written (GB)	Data Read (GB)	Data Written (GB)
3	1 TB	95%	3.03	0.00	1.74	1.15
4		37%	2.58	0.00	703.00	702.00
30		91%	0.32	0.00	0.00	0.00
3	3 TB	97%	9.44	0.00	5.50	2.48
4		21%	8.28	0.00	9512.00	9521.00
30		34%	0.00	0.00	3920.00	3924.00

Fig. 5. Processor/Disk Characteristics of worst scaling SQL queries - 3, 4, 30

described in Sect. 1 [5,6,8,9]. Floratou et al. [1] study the performance differences of two SQL engines - Hive and Impala against data stored in Hadoop using TPC-H and TPC-DS based workloads. Chen et al. [3] present a performance evaluation of five SQL-on-Hadoop solutions using a subset of TPC-DS queries. Similarly, Poggi et al. [2] present results of a performance study characterizing SQL-on-Hadoop solutions provided by four major cloud providers using TPC-H.

All of these contributions study big data systems but none of them uses BigBench which includes a broader set of processing paradigms (machine learning, natural language processing) and variety of data formats including semi-structured (web-clickstreams) and unstructured data (product reviews) data. Also, BigBench has support for up to petabyte scale databases while TPC-H or TPC-DS are limited to 100 TB. The current publications of TPC-H/DS concentrate primarily on 10 TB databases while for BigBench we have a result with 30 TB of data as well [19,37].

Next, we summarize few research studies that have used BigBench over different frameworks to evaluate performance of hardware and software components of their systems. In a recent study in 2017, Poggi et al. [11] complemented their prior work by looking at BigBench to characterize Spark and Hive performance across different PaaS (Platform-as-a-Service) offerings. There have been few studies and official submissions at TPC that employ BigBench to analyze and compare performance of Big Data offerings [18,19]. But all of these evaluations are over cluster-based environments. We are not aware of any studies that have explored BigBench on a single node configuration and a relational database engine such as Microsoft* SQL Server*. Wang et al. [12] share results on running BigBench on a 9-node Cloudera cluster. Their experiments are designed to understand the impact of core and disk scaling and tuning CPU frequency on query response time. While we are yet to do similar studies, our preliminary results show how queries 2, 4, and 30 are impacted by disk bandwith since we observe spills to Tempdb for each of these queries (primarily queries 4 and 30). Cao et al. [13] present results over a cluster-based configuration (two cluster setups - 9 node, 60 node) running

BigBench for Hive over MapReduce. Complete run of 30 queries on their 9-node cluster (total of 216 cores and 2.2 TB memory) takes 34,076 s while 3 TB run on our system takes 44,376 s. Now, we do observe optimization opportunities which if implemented can improve the total runtime and the core count on the cluster setup is almost double of what we have on our system. The paper does not show individual query times which would have enabled us to compare each query's performance. CPU intensive queries siuch as query 3 would significantly benefit from higher number of cores. Baru et al. [10] looked at 1 TB scale factor over a 6-node cluster configuration with a total of 120 cores and 768 GB main memory but their total runtime is still much higher than our system. Authors have mentioned that it is raw, out-of-the-box performance and the runtimes could potentially improve with tuning. The study done by Ivanov et al. [18] presents detailed analysis and results on running BigBench over a 4-node cluster using Hive and Spark frameworks and across multiple scale factors. Given that the configuration had a total of 128 GB memory for the 1 TB BigBench scale factor, it is not practical to compare their runtimes with our results. The authors also present resource utilization of a subset of queries and it would be worthwhile to study it and identify if there are any insights that can be gained about the core characteristics of the workload from it.

On our third survey dimension identifying contributions which have looked at benchmarking single node SQL Server*, the evaluations have been based on TPC-H or TPC-DS [4,20,21,37]. Our work is the first attempt to measure and demonstrate performance of Microsoft* SQL Server* 2019 (pre release) on the Intel® Xeon™ processor 8164 product family using BigBench as the workload.

6 Conclusions and Future Work

This paper presents our experiences and initial experiments using BigBench on a single node configuration powered by Intel technologies and a relational database system, Microsoft* SQL Server*. Our initial results on 1 and 3 TB data sizes demonstrate advanced capabilities of Microsoft SQL Server 2019 (pre-release candidate) to handle heterogeneous and volume aspects of big data and how even a single-node, relational database configuration can scale up to big data workloads.

Given that this paper is an early study, there exists several avenues for future research. Firstly, collecting and analyzing performance over higher scale factors which are even more representative of the data volume aspect in big data is an ongoing study. Secondly, profiling the benchmark to assess sensitivities of BigBench queries to the number of cores, core frequency, memory, and storage in a single node environment is another promising direction. There are similar studies done over cluster-based environments. Combined with the existing studies on cluster-based configurations, these results can be used by practitioners to compare the query resource requirements and processing methodology in a single vs. multi-node configuration, and thus understand the impact of these different architectures on the performance of big data workloads. Also, it would be

important to identify optimal platform configuration settings since the current configuration may have been overconfigured for the scale factors considered in this study. Another interesting direction would be to expand analysis to address multiple concurrent streams. Richins et al. [23] have done a comprehensive analysis using BigBench on a cluster-based configuration. The authors have identified thread level parallelism as a major bottleneck. It would be worthwhile to investigate if similar behaviour shows up on single-node setup as well and drive further analysis based on the results.

Acknowledgements. We thank Harish Sukhwani, Mahmut Aktasoglu, Hamesh Patel from Intel, and Jasraj Dange and Tri Tran from Microsoft Corporation for their constructive feedback that helped to improve the paper. We are immensely grateful to Nellie Gustafsson from Microsoft for her help in revising machine learning queries to match the benchmark specification. We thank Arun Gurunathan, Sumit Kumar, Nellie Gustafsson, and Gary Ericson for their inputs on revising the section on extensibility framework. The authors would also like to acknowledge Vaishali Paliwal and Charles Winstead from Intel Corporation for their overall support and project guidance, Sridharan Sakthivelu for the technical discussions, and Ketki Haridas for her early contributions to the work.

Appendix

In this section we present two BigBench queries (10, 20) to highlight SQL Server capabilities towards implementing queries that use machine learning algorithms and third-party libraries such as Java. Query 10 is a sentiment analysis query that uses java libraries and query 20 employs a k-means clustering algorithm in machine learning. While we used SQL Server* CTP 2.4 version for our experiments, the code shown here is based on the latest CTP 3.0 version to highlight the latest feature implementations supported by Microsoft at the time of writing of this paper [40].

Query 10 - Using Java based libraries and user code

```
/*
Query description
For all products, extract sentences from its product reviews that
    contain positive or negative sentiment and display for each item the
    sentiment polarity of the extracted sentences (POS OR NEG) and the
    sentence and word in sentence leading to this classification*/

CREATE OR ALTER PROCEDURE [dbo].[query10] @param1 nvarchar(20), @param2
    nvarchar(15) , @param3 bigint, @param4 nvarchar(50), @param5
    nvarchar(20), @query nvarchar(400)
AS
BEGIN
-- Saving query results in a table
drop table if exists Q10Results;
Create table Q10Results
```

```
(
itemid bigint
,sentence varchar(4000)
,sentiment varchar(3)
,token varchar(20)
)
--The method invoked in the Java code is always the "execute" method
EXEC sp_execute_external_script
  @language = N'Java'
, @script = N'Query10.SentimentAnalysis'
, @input_data_1 = @query
, @params = N'@tablename nvarchar(20), @serverInstanceName nvarchar(15)
    , @port bigint, @modelsParentFolder nvarchar(50), @databaseName
    nvarchar(20)'
, @tablename = @param1
, @serverInstanceName = @param2
, @port= @param3
, @modelsParentFolder= @param4
, @databaseName= @param5
with result sets ((itemsk bigint, sentence varchar(4000), sentiment
    varchar(3), word varchar(20)));
END
GO

--Now execute the above stored procedure and provide the input
    parameters and an input query
EXECUTE [dbo].[query10] N'Q10Results', N'SQL2019CTP3', 11212,
    N'C:\NLPSQLCTP25', N'TPCxBB_1GB_2019', N'SELECT pr_item_sk,
    pr_review_content FROM product_reviews option(maxdop 20)'
GO
END
```

Query 20 - Using machine learning k-means algorithm

```
DROP PROCEDURE IF EXISTS [q20_create_customer_return_clusters]
GO
CREATE PROCEDURE [dbo].[q20_create_customer_return_clusters]
AS
/*
This procedure uses R to classify customers into different groups based
    on their purchase & return history.

Query description
Customer segmentation for return analysis: Customers are separated along
    the following dimensions: return frequency, return order ratio
    (total number of orders partially or fully returned versus the total
    number of orders), return item ratio (total number of items returned
    versus the number of items purchased), return amount ration (total
    monetary amount of items returned versus the amount purchased),
```

```
    return order ratio. Consider the store returns during a given year
    for the computation. */
BEGIN

DECLARE @model_generation_duration float
    , @predict_duration float
    , @instance_name nvarchar(100) = @@SERVERNAME
    , @database_name nvarchar(128) = DB_NAME()

-- Input query to geenerate the purchase history & return metrics
    , @input_query nvarchar(max) = N'
SELECT
  ss_customer_sk AS customer,
  ROUND(COALESCE(returns_count / NULLIF(1.0*orders_count, 0), 0), 7) AS
      orderRatio,
  ROUND(COALESCE(returns_items / NULLIF(1.0*orders_items, 0), 0), 7) AS
      itemsRatio,
  ROUND(COALESCE(returns_money / NULLIF(1.0*orders_money, 0), 0), 7) AS
      monetaryRatio,
  COALESCE(returns_count, 0) AS frequency
FROM
  (
    SELECT
      ss_customer_sk,
      -- return order ratio
      COUNT_BIG(DISTINCT ss_ticket_number) AS orders_count,
      -- return ss_item_sk ratio
      COUNT_BIG(ss_item_sk) AS orders_items,
      -- return monetary amount ratio
      SUM( ss_net_paid ) AS orders_money
    FROM store_sales s
    GROUP BY ss_customer_sk
  ) orders
  LEFT OUTER JOIN
  (
    SELECT
      sr_customer_sk,
      -- return order ratio
      COUNT_BIG(DISTINCT sr_ticket_number) as returns_count,
      -- return ss_item_sk ratio
      COUNT_BIG(sr_item_sk) as returns_items,
      -- return monetary amount ratio
      SUM( sr_return_amt ) AS returns_money
    FROM store_returns
    GROUP BY sr_customer_sk
  ) returned ON ss_customer_sk=sr_customer_sk
 '

 EXECUTE sp_execute_external_script
         @language = N'R'
```

```
        , @script = N'
# Define the connection string
connStr <- paste("Driver=SQL Server;Server=", instance_name,
    ";Database=", database_name, ";Trusted_Connection=true;", sep="");
cc <- RxInSqlServer(connectionString=connStr)
rxSetComputeContext(cc)

customer_returns <- RxSqlServerData(sqlQuery = input_query, colClasses =
    c(customer = "numeric", orderRatio = "numeric",
                                itemsRatio = "numeric", monetaryRatio =
                                    "numeric", frequency =
                                    "numeric"),connectionString =
                                    connStr);

# Output table to hold the customer group mappings
return_cluster = RxSqlServerData(table = "customer_return_clusters",
    connectionString = connStr);

# set.seed for random number generator for predictability
set.seed(10);

# generate clusters using rxKmeans and output key / cluster to a table
model_generation_duration <- system.time(clust <- rxKmeans( ~ orderRatio
    + itemsRatio + monetaryRatio + frequency, customer_returns,
    numClusters = 10
                , outFile = return_cluster, outColName = "cluster",
                    extraVarsToWrite = c("customer"), overwrite =
                    TRUE))[3];
'
    , @input_data_1 = N''
    , @params = N'@instance_name nvarchar(100), @database_name
        nvarchar(128),@input_query nvarchar(max),
        @model_generation_duration float OUTPUT'
    , @instance_name = @instance_name
    , @database_name = @database_name
    , @input_query=@input_query
    , @model_generation_duration = @model_generation_duration OUTPUT;

    PRINT CONCAT(N'Model generation time: ', @model_generation_duration,
        ' seconds.');

END
GO
exec [dbo].[q20_create_customer_return_clusters]
```

References

1. Floratou, A., Minhas, U.F., Özcan, F.: SQL-on-Hadoop: full circle back to shared-nothing database architectures. Proc. VLDB Endowment **7**(12), 1295–1306 (2014)
2. Poggi, N., et al.: The state of SQL-on-Hadoop in the Cloud. In: 2016 IEEE International Conference on Big Data (Big Data), pp. 1432–1443. IEEE, December 2016
3. Chen, Y., et al.: A study of SQL-on-hadoop systems. In: Zhan, J., Han, R., Weng, C. (eds.) BPOE 2014. LNCS, vol. 8807, pp. 154–166. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13021-7_12
4. Larson, P.A., et al.: Enhancements to SQL server column stores. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, pp. 1159–1168. ACM, June 2013
5. Ivanov, T., et al.: Big data benchmark compendium. In: Nambiar, R., Poess, M. (eds.) TPCTC 2015. LNCS, vol. 9508, pp. 135–155. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-31409-9_9
6. Han, R., John, L.K., Zhan, J.: Benchmarking big data systems: a review. In: IEEE Transactions on Services Computing, pp. 580–597 (2017)
7. Ghazal, A., et al.: Bigbench V2: the new and improved bigbench. In: 2017 IEEE 33rd International Conference on Data Engineering (ICDE), pp. 1225–1236. IEEE, April 2017
8. https://github.com/c9n/hadoop/blob/master/hadoop-mapreduce-project/hadoop-mapreduce-client/hadoop-mapreduce-client-jobclient/src/test/java/org/apache/hadoop/hdfs/NNBench.java . Accessed 29 May 2019
9. https://www.ibm.com/support/knowledgecenter/en/SSGSMK_7.1.1/mapreduce_integration/map_reduce_terasort_example.html . Accessed 29 May 2019
10. Baru, C., et al.: Discussion of bigbench: a proposed industry standard performance benchmark for big data. In: Nambiar, R., Poess, M. (eds.) TPCTC 2014. LNCS, vol. 8904, pp. 44–63. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-15350-6_4
11. Poggi, N., Montero, A., Carrera, D.: Characterizing bigbench queries, hive, and spark in multi-cloud environments. In: Nambiar, R., Poess, M. (eds.) TPCTC 2017. LNCS, vol. 10661, pp. 55–74. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-72401-0_5
12. Wang K., Bian B., Cao P., Riess M. https://www.intel.com/content/dam/www/public/.../cofluent-tpcx-bb-express-paper.pdf. Accessed 1 June 2019
13. Cao, P., et al.: From BigBench to TPCx-BB: standardization of a big data benchmark. In: Nambiar, R., Poess, M. (eds.) TPCTC 2016. LNCS, vol. 10080, pp. 24–44. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-54334-5_3
14. TPC-DS, http://www.tpc.org/tpcds/. Accessed 29 May 2019
15. Big Data Benchmark for BigBench. https://github.com/intel-hadoop/Big-Data-Benchmark-for-Big-Bench. Accessed 29 May 2019
16. Big data: The next frontier for innovation, competition, and productivity. https://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/big-data-the-next-frontier-for-innovation. Accessed 29 May 2019
17. Windows Performance Monitor Overview. https://techcommunity.microsoft.com/t5/Ask-The-Performance-Team/Windows-Performance-Monitor-Overview/ba-p/375481. Accessed 29 May 2019
18. Ivanov, T., Beer, M.G.: Evaluating hive and spark SQL with BigBench. arXiv preprint arXiv:1512.08417 (2015)

19. TPCx-BB - Top Ten Performance Results. http://www.tpc.org/tpcx-bb/results/tpcxbb_perf_results.asp. Accessed 29 May 2019
20. TPC-DS Top Performance Results. http://www.tpc.org/tpcds/results/tpcds_perf_results.asp. Accessed 29 May 2019
21. Sen, R., Ramachandra, K.: Characterizing resource sensitivity of database workloads. In: 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 657–669. IEEE, February 2018
22. View Resource Governor Properties. https://docs.microsoft.com/en-us/sql/relational-databases/resource-governor/view-resource-governor-properties?view=sql-server-2017. Accessed 29 May 2019
23. Richins, D., Ahmed, T., Clapp, R., Reddi, V.J.: Amdahl's law in big data analytics: alive and kicking in TPCx-BB (BigBench). In: 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 630–642. IEEE, February 2018
24. Huppler, K., Johnson, D.: TPC express – a new path for TPC benchmarks. In: Nambiar, R., Poess, M. (eds.) TPCTC 2013. LNCS, vol. 8391, pp. 48–60. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-04936-6_4
25. Apache Hadoop. https://hadoop.apache.org/. Accessed 30 May 2019
26. Apache Spark. https://spark.apache.org/. Accessed 30 May 2019
27. Apache Cassandra. http://cassandra.apache.org/. Accessed 30 May 2019
28. Apache Hive. https://hive.apache.org/. Accessed 30 May 2019
29. Microsoft Azure HDInsight. https://azure.microsoft.com/en-us/services/hdinsight/. Accessed 30 May 2019
30. Teradata Aster. https://downloads.teradata.com/aster. Accessed 30 May 2019
31. Bakshi, K.: Considerations for big data: architecture and approach. In: 2012 IEEE Aerospace Conference (2012)
32. Jia, Z., et al.: Characterizing and subsetting big data workloads. In: 2014 IEEE International Symposium on Workload Characterization (IISWC), pp. 191–201. IEEE, October 2014
33. Malik, M., Rafatirad, S., Homayoun, H.: System and architecture level characterization of big data applications on big and little core server architectures. ACM Trans. Model. Performance Eval. Comput. Syst. (TOMPECS) **3**(3), 14 (2018)
34. Madden, S.: From databases to big data. IEEE Internet Comput. **16**(3), 4–6 (2012)
35. Chen, J., et al.: Big data challenge: a data management perspective. Front. Comput. Sci. **7**(2), 157–164 (2013)
36. Santos, M.Y., et al.: Evaluating SQL-on-Hadoop for big data warehousing on not-so-good hardware. In: Proceedings of the 21st International Database Engineering & Applications Symposium, pp. 242–252. ACM, July 2017
37. TPC-H Top Performance Results. http://www.tpc.org/tpch/results/tpch_perf_results.asp. Accessed 30 May 2019
38. Extensibility architecture in SQL Server Language Extensions. https://docs.microsoft.com/en-us/sql/language-extensions/concepts/extensibility-framework?view=sqlallproducts-allversions. Accessed 29 July 29 2019
39. RevoScaleR package. https://docs.microsoft.com/en-us/machine-learning-server/r-reference/revoscaler/revoscaler. Accessed 11 June 2019
40. What new in SQL Server Language Extensions?. https://docs.microsoft.com/en-us/sql/language-extensions/language-extensions-whats-new?view=sqlallproducts-allversions. Accessed 11 June 2019
41. Resource governance for machine learning in SQL Server. https://docs.microsoft.com/en-us/sql/advanced-analytics/administration/resource-governance?view=sql-server-2017. Accessed 13 June 2019

42. Ghazal, A., et al.: BigBench: towards an industry standard benchmark for big data analytics. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, pp. 1197–1208. ACM, June 2013
43. Rabl, T., et al.: BigBench specification V0.1. In: Rabl, T., Poess, M., Baru, C., Jacobsen, H.-A. (eds.) WBDB -2012. LNCS, vol. 8163, pp. 164–201. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-53974-9_14