Benchmarking Elastic Cloud Big Data Services under SLA Constraints

Nicolas Poggi*[1,2], Víctor Cuevas-Vicenttín[1], Josep Lluis Berral[1,2], Thomas Fenech[1], Gonzalo Gómez[1], Davide Brini[1], Alejandro Montero[1], David Carrera[1,2]

Umar Farooq Minhas[3], Jose A. Blakeley[3], Donald Kossmann[3], Raghu Ramakrishnan[3], and Clemens Szyperski[3]

[1] Barcelona Supercomputing Center (BSC) firstname.lastname@bsc.es
[2] Universitat Politècnica de Catlalunya (UPC - BarcelonaTech)
[3] Microsoft Corporation, Microsoft Research (MSR)
{ufminhas,joseb,donaldk,raghu,clemens}@microsoft.com

Abstract. We introduce an extension for TPC benchmarks addressing the requirements of big data processing in cloud environments. We characterize it as the Elasticity Test and evaluate under TPCx-BB (BigBench). First, the Elasticity Test incorporates an approach to generate real-world query submissions patterns with distinct data scale factors based on major industrial cluster logs. Second, a new metric is introduced based on Service Level Agreements (SLAs) that takes the quality of service requirements of each query under consideration.
Experiments with Apache Hive and Spark on the cloud platforms of three major vendors validate our approach by comparing to the current TPCx-BB metric. Results show how systems who fail to meet SLAs under concurrency due to queuing or degraded performance negatively affect the new metric. On the other hand, elastic systems meet a higher percentage of SLAs and thus are rewarded in the new metric. Such systems have the ability to scale up and down compute workers according to the demands of a varying workload and can thus save dollar costs.

Keywords: benchmarking · big data · databases · cloud · SLA · QoS.

1 Introduction

BigBench [1] was standardized by the Transaction Processing Performance Council (TPC) as TPCx-BB. It is the first reference and implemented benchmark for Big Data Analytics. BigBench addresses important aspects in Big Data such as variety (considering structured and semi-structured data), volume (through large and varying scale factors), and velocity (through a policy of interspersing inserts, deletes, and updates with query streams).

However, BigBench does not fully address some of the aspects relevant to data analytics services operating in cloud infrastructures. For instance, the benchmark runs for a single database size tied to a scale factor, while in real-life, we can expect queries processing varying amounts of data at the same time. Furthermore,

* Contribution while at the BSC-MSR Centre, currently at Databricks Inc.

in BigBench queries are run in n concurrent streams (where n is defined by the user), while the sequence of queries to be executed for each particular stream is specified in a placement table. Thus, the system load is constant in terms of the number of queries under execution e.g., queries start only when the previous from the stream has finished and do not queue in the system.

Such an approach is adequate to replicate a homogeneous workload with aggregate performance considerations. It is inappropriate, however, when the workload is expected to vary with time in response to real user demands, and when users can have specific performance or isolation expectations for each query i.e., for batch, interactive, and streaming queries. Moreover, elastic services such of a database or query as-a-service (DBaaS or QaaS) provide the ability to scale up or down compute resources to either process more rapidly or to save costs i.e., in periods of low intensity. We address these problems by introducing a new test, which we call the Elasticity Test and incorporating it into the existing TPCx-BB benchmark. We test the approach in TPCx-BB here, while the test could be applied directly to other TPC benchmarks, in particular, TPC-DS and TPC-H.

Our extension is built from three main components. First, we design and implement a workload generation approach that produces workloads that are more representative of what we can expect in a highly dynamic environment such as the cloud (Section 2). Second, we implement a driver capable of executing such workloads. Third, we propose a new benchmark metric based on Service Level Agreements (SLAs) (Section 3), which enables to measure the compliance with the performance expectations of the user and thus the quality of service (QoS) of the system under test.

In Sections 4 and 5, we present and analyze the results of experiments that use our extended TPCx-BB benchmark to evaluative Apache Hive and Spark running on three major cloud platforms. Subsequently, we present related work and the conclusions.

2 Workload characterization and generation

2.1 Analyzing Cloud services workloads

We present how we generate realistic workloads based on the analysis of data obtained from a real-world production cluster, the Cosmos cluster [4] operated at Microsoft. The dataset analyzed consists of about 350,000 job submissions sampled from the period between January 2 to February 9, 2017.

Modeling the arrival rate To implement a benchmark that reflects the workloads that real-world Big Data computing infrastructures face, a fundamental aspect is to understand the arrival rates of jobs, i.e. the number of jobs that arrive for each unit of time, and to model it adequately. With this purpose in mind, we analyzed the job submissions in the Cosmos sample dataset.

Identifiable trends in the arrival rate of jobs, such as peaks in usage during working hours and a decline in activity on the weekends, led us to conclude that the periodicity and variability of the job submission rate cannot be captured through a simple approach. For instance, if we were to define a single cumulative distribution

function (CDF), this CDF would conflate all of the data irrespective of the particular patterns observed. Instead we need to adopt a more sophisticated approach capable of temporal pattern recognition.

A technique capable of modeling temporal patterns in the arrival rate is Hidden Markov Models (HMMs). A HMM enables us to take into account the transition between busy and quiet periods. In practice, the arrival rate varies continuously, but it can be modeled using a series of n discrete levels associated with states. At a given point in time, the arrival rate is represented by a series of parameters, describing the probability of being in each level (1 to n), along with the probabilities of transitioning to each of the other levels. In our case, we found that a model consisting of four levels was sufficient to capture the fluctuations in activity over the period in the dataset.

Once the model has been trained with the reference data, it can be used to produce a synthetic randomized sequence of job arrivals, which possesses similar statistical characteristics to the reference data. The generated arrival sequence can then be scaled in terms of the range of arrival rates and the period, which makes the model a useful part of a realistic workload simulator. The output model is expected to have periods of a high number of job submissions (*peaks*), as well as periods of low (*valleys*) or even no submissions where the system might be *idle*.

To validate the capability of the model to produce an arrival rate whose properties match those of the reference dataset, but adding variability and avoiding being fully deterministic, we can generate synthetic sequences of job submissions and then compare their distributions. Example synthetic distributions of arrival rates are shown in Figure 1. Kernel density estimation (KDE) is used to approximate the distribution of each data series. These data series correspond to the generated job arrival data with the original states and with newly generated states (a distinction explained below). KDE is a non-parametric technique to estimate probability density functions that yields a result closely related to histograms, hence it is an adequate technique to perform this comparison. The lines are close to one another but not equal, indicating a close match between the sequences generated and a degree of non-determinism.

The aforementioned HMM is encapsulated in a script that receives the following parameters: *dataset*, *max jobs*, minutes/hours, keep the original state sequence. The dataset parameter corresponds in our experiments to the Cosmos dataset. The *max jobs* parameter denotes the maximum number of jobs that can be generated for each time unit, which in turn is defined to be in seconds (or fractions of a second) by the third parameter. A sequence of states has to be generated by the algorithm to produce the HMM, this same sequence can then be used to produce the output data, or alternatively, a new sequence can be generated. This is specified by the last parameter that takes a boolean value.

Input data size Another fundamental aspect of modeling a realistic workload is the complexity of the submitted jobs. In this paper we address data complexity, related to the size of the database; whereas query type or complexity, related to the query expression, forms part of future work.

We investigated the presence of temporal and periodic effects on the size of the job input data sizes, which can occur in particular cases. For example, we

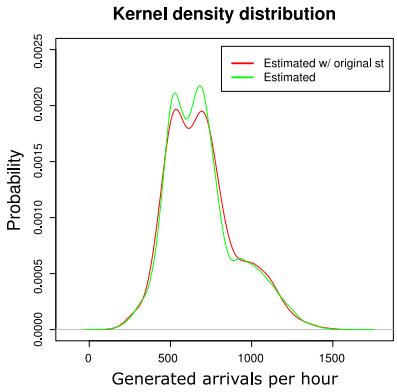


Fig. 1: Comparison between two arrival rate sequences generated by the model.

can expect that some users will prefer to run numerous short jobs during business hours followed by a few data-intensive jobs during the night shift. However, we did not find evidence that such a pattern holds across all of the data. Therefore, while the job arrival rate varies significantly, as stated earlier, the distribution of the corresponding input data sizes is not subject to similar temporal patterns. Consequently, we take an aggregate view of job input data sizes. For benchmarking, the effects of our analysis are reflected only on the scale factors that the BigBench queries are subject to.

We present the cumulative distribution function for the input data sizes of the sampled Cosmos dataset in Figure 2; it is a discrete CDF with the data sizes axis expressed in log scale and with data points corresponding to the usual units of measuring data. The size of the input data varies from a few bytes to up to near a petabyte. Almost 60% of the jobs have less than 1 GB of data as input and very few more than 10 TB. Recall that we are dealing with sampled data, so such outliers may not be representative of the full dataset.

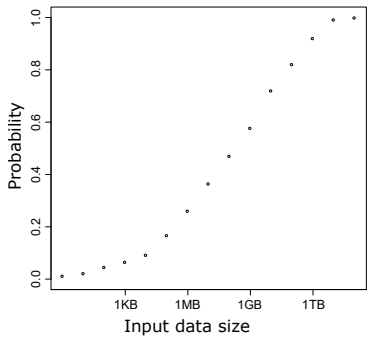


Fig. 2: CDF of the input data sizes of the jobs in the Cosmos sample dataset.

From the CDF we can generate random variates representing the input data sizes of the jobs in the workload simulation. The generated input data sizes are then matched to scale factors in the BigBench database. We consider only a few representative scale factors in factors of 10 i.e., 1 GB, 100 GB, 1 TB, 10 TB.

We next show how our model derived from the analysis of the Cosmos dataset enables us to generate a workload suitable for our BigBench extension, experimental results for this workload are presented in Section 4.

2.2 Workload generation for Cloud services

Our goal for the Elasticity Test is to create a realistic workload that can be integrated with existing TPC metrics. We explain this integration to TPCx-BB in Section 3, although the Elasticity Test can also be run in stand-alone manner. With that goal in mind, we adapted the notion of streams employed in TPC benchmarks, where for a given number of streams n, one instance of each query appears in the stream. Thus, the number of streams defines the total number of queries in the workload, namely, n times the number of different queries in the benchmark.

Once the total number of queries in the workload is defined, these are assigned randomly to time-slots (batches) whose size is determined by the HMM, subject to a maximum $n-1$ streams. Setting the maximum limit to $n-1$ makes the Elasticity Test compatible with the Throughput Test, as it is expected that n is a user-defined threshold where the system is expected to complete all the queries. Also, it prevents introducing a new parameter into the benchmark, simplifying adoption. Repetition of queries is not allowed within a single batch but can occur among different batches. The query batches can be viewed as the static specification of the workload, which is then made dynamic during execution by a batch interval, λ_{batch} , optionally defined by the user and that determines the time interval between the submissions of each batch of queries (defaults to 60 s.). Having a smaller value of λ_{batch} means that the driver will submit the batches faster i.e., put more load on the System Under Test (SUT), while a larger value results in longer waiting times between query batches. Having a lower interval could produce lower test times to improve the final score if the system is able to run the queries without SLA-penalties.

For the experiments, as we increase the data scale factor from 1 TB to 10 TB, to compensate for the corresponding increase in system load, the number of streams is lowered. Specifically, we use: from $n = 4$ at 1 TB we lower the number of streams to $n = 2$ at 10 TB. We also increase the batch interval λ_{batch}, to account for the higher time taken to complete queries for larger data sizes as the systems failed to meet the SLAs with lower values.

At 1 TB, the above parameters result in the workload presented in Figure 3 (left). The blue line on the chart (left axis) shows the total number of queries that have been submitted to the system. The green impulses (right axis) show the size of each batch of jobs, up to a maximum of 3 (always less than or equal to $n = 4$). The sizes of the batches increase at around half of the timeline, implying that a statically-provisioned system with sufficient capacity to handle the maximum batch size is over-provisioned for the first part of the test, so the total cost of the execution may be higher than that of an elastic system.

For the workload in Figure 3 (left) we consider only the 14 SQL queries from BigBench, which means that the total number of queries is $n \cdot M = 56$, reached at

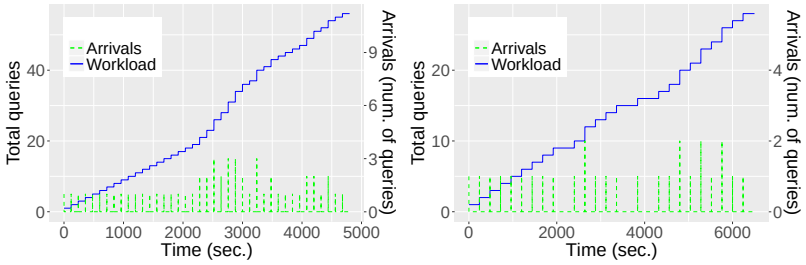


Fig. 3: Workloads for the 1 TB (left) and 10 TB (right) scale factors.

the top right corner of the graph. We note that although in this workload all of the queries that form part of a batch arrive at the same time, our driver provides the option of evenly distributing the queries of every individual batch across their corresponding batch interval. In that case, we would observe much lower green lines (of height 1, in fact), but there would be a much higher number of them in the chart.

Figure 3 (right) depicts the workload for the 10 TB scale factor. With the number of streams being reduced to $n = 2$, the only possible batch sizes are 0, 1 and 2. The batch interval λ_{batch} is 240 seconds and the total number of queries is 28. Note that some batches have zero queries, representing the low-workload periods that occur in systems like Cosmos.

For dynamic workloads as exemplified above, it is highly desirable to measure not only the query execution times, but also how well these execution times meet the user requirements, particularly when the system is subject to higher loads. For this purpose, we introduce in the next section a SLA-aware performance metric.

3 New SLA-aware metric for Big Data systems

A shortcoming of the current TPCx-BB specification is that the quality of service (QoS) per query is not measured. This means that a system can achieve a high score by completing parts of the execution quickly while neglecting others. In a real-world business use case, workloads have deadlines and an expected QoS, so a system operating at scale must be able to maintain a reasonable response time for individual queries.

In this section, we present a new Elasticity Metric to be incorporated into TPCx-BB, which assesses the ability of a system to deliver a consistent quality of service at a per-query level. The metric is based on the concept of each query having a completion deadline, i.e., a Service Level Agreement (SLA). We define a per-query SLA, and to achieve the maximum score, the system must execute each query within its SLA. The system is penalized for the number of queries that fail to meet their SLA, and by how much time the deadline was missed. Recall that in the Elasticity Test, we submit queries to the system according to fluctuating batch arrivals, generated using the HMM model presented in Section 2.1, which can produce a queuing effect where more queries are running concurrently than in the previous Throughput Test.

We begin this section by giving a brief overview of the performance metric used in TPCx-BB. We then describe how to calculate the score corresponding to

our proposed metric. Finally, we conclude this section by proposing a means of integrating the Elasticity Test into the TPCx-BB.

Existing performance metric We first revisit the existing formula used to calculate the score in TPCx-BB. The overall Performance Metric is defined as

$$BBQpm@SF = \frac{SF \cdot 60 \cdot M}{T_{LD} + \sqrt[2]{T_{PT} \cdot T_{TT}}} \tag{1}$$

The three variables T_{LD}, T_{PT} and T_{TT} are derived, as described below, from the time taken to run the Load Test, Power Test, and Throughput Test, respectively. SF is the scale factor and M is the total number of queries in the benchmark. The Performance Metric is designed to represent the number of queries per minute that a system can execute, which decreases as the various test times increase. The square root of $T_{PT} \cdot T_{TT}$ is the geometric mean of the two values, which is more suitable than the more commonly-used arithmetic mean, in situations where values are associated with different scales.

The Load Test measures the time taken to prepare the data for the benchmark. The details of this test vary, depending on the SUT, but the total time T_{Load} is multiplied by a constant factor of 0.1 to reduce the weight of this score, i.e.,

$$T_{LD} = 0.1 \cdot T_{Load} \tag{2}$$

The Power Test measures the raw performance of the system, running the full suite of queries serially. It is calculated as follows

$$T_{PT} = M \cdot \sqrt[M]{\prod_{i=1}^{M} t_i} \tag{3}$$

Where t_i is the time taken to execute query i, measured in seconds.

The Throughput Test measures the ability of the system to execute multiple streams in parallel, defined as

$$T_{TT} = \frac{1}{n} \cdot T_{Tput} \tag{4}$$

where n is the total number of streams, chosen by the user, and T_{Tput} is the elapsed time between the start of the Throughput Test and the completion of all streams, measured in seconds.

New performance metric Complementing the existing T_{LD}, T_{PT} and T_{TT}, we introduce a new performance metric, T_{ET}, which is defined as

$$T_{ET} = \lambda_{batch} \cdot \Delta_{SLA} \cdot \rho_{SLA} \cdot T_{el} \tag{5}$$

Each of the terms is described below. In all cases, n refers to the maximum level of concurrency during the test. By default, the value of n used in the Elasticity Test will be the same as the total number of streams used in the Throughput Test. The number of streams is used to reduce the number of user-defined parameters in the new test, also due to n usually corresponding to the maximum concurrency of the SUT and scale factor intended by the user. Again, M is the number of queries in the benchmark.

- λ_{batch} is the time interval between job batch submissions. This is the only additional optional parameter for the user. Defaults to 60 s.
- Δ_{SLA} is the SLA distance, which is the average failure ratio of queries that do not complete within their SLA, and is defined as

$$\Delta_{SLA} = \frac{1}{n \cdot M} \cdot max\left(1, \sum_{i=1}^{n \cdot M} max\left(0, \frac{t_i - SLA_{Q(i)}}{SLA_{Q(i)}}\right)\right) \tag{6}$$

  where t_i is the time taken to run query i from the schedule and $SLA_{Q(i)}$ is the SLA for query i. The inner max ensures that queries which complete within their SLA do not contribute to the sum. The outer max means that when all queries pass their SLA, $\Delta_{SLA} = 1$.
- ρ_{SLA} is the *SLA factor*, which is defined as

$$\rho_{SLA} = \frac{1}{n \cdot M} \cdot max\left(1, \frac{N_{fail}}{0.25}\right) \tag{7}$$

  where N_{fail} is the number of queries that fail to meet their SLA. $\rho_{SLA} < 1$ when under 25% of the queries fail their SLAs, which reduces (improves) the value of the metric T_{ET}. Conversely, failing more than 25% of the queries results in $\rho_{SLA} > 1$, which has a negative impact on T_{ET}. As in the definition of Δ_{SLA}, a max is applied to prevent ρ_{SLA} from reaching zero, limiting the minimum value to $1/(n \cdot M)$.
- T_{el} is the total elapsed time of the Elasticity Test.

The score from the Elasticity Test is incorporated into the existing TPCx-BB formula as follows

$$BB++Qpm@SF = \frac{SF \cdot 60 \cdot M}{T_{LD} + \sqrt[3]{T_{PT} \cdot T_{TT} \cdot T_{ET}}} \tag{8}$$

That is, the geometric mean is extended to include the score from the Elasticity Test. Since the range of scores in the Power Test, the Throughput Test, and the Elasticity Test are all different, scores obtained using this new metric cannot be compared directly with older results. However, it is possible to calculate the new metric using the existing Load Test, Power Test and Throughput Test results, provided that the Elasticity Test is run with the same number of streams n. The next section presents experiments in which we apply this new metric.

4 Experimental evaluation

We now present the results of executing the workloads derived from our model based on the analysis of the Cosmos dataset and evaluated by our SLA-based metric. As discussed in Section 3, the computation of our benchmark score requires, in addition to the Elasticity Test metrics, also the metrics of the current BigBench tests; namely for the Data Loading, the Power, and the Throughput tests. We present the results of our experiments in regards to each of these next. Due to failures and scalability problems of the non-SQL queries, all the presented experiments were limited to the 14 SQL-only queries.

The SUTs considered are Apache Hive and Apache Spark running on three major cloud providers. We avoid identifying the particular providers and refer to them only through pseudonyms: ProvA, ProvB, and ProvC. On the three providers, Hive was either version 2.2 or 2.3 and Spark with 2.1 or 2.2.

We considered 1 TB and 10 TB scale factors, although not all configurations for the latter. In some cases, failures at a particular test prevent us from presenting the corresponding test scores and also to compute the global metrics. The configurations of the different SUTs were defined in such a way that they include 32 worker nodes, with 16 virtual cores (512-vcpus in total), and 64 GB of RAM each (2 TB total).

4.1 Data Loading Test

We present the measured loading times for the 1 TB scale factor in Figure 4 (left). Hive outperforms Spark in ProvA and ProvB, but the opposite is true for ProvC. Also, ProvB and ProvC achieve lower loading times than ProvA. We note that for Hive and Spark the data were stored using the ORC columnar storage format.

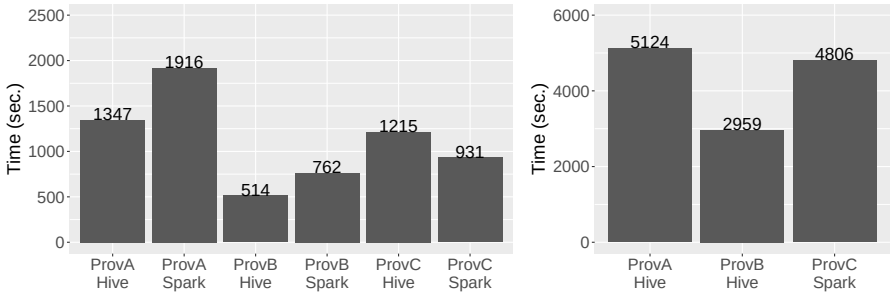


Fig. 4: Data loading times for the 1 TB (left) and 10 TB (right) scale factors.

The second scale factor that we considered is 10 TB, which involved only some of the SUTs and whose results are illustrated in Figure 4 (right). The data loaded is used for both Hive and Spark. The lowest load time overall corresponds to Hive on ProvB, followed by Spark on ProvC, whose value is close to the results of ProvA.

4.2 Elasticity Test

An extension of the TPCx-BB driver was required to execute the Elasticity Test. The structure of the driver we implemented enables the execution of queries by independent threads and at various data sizes within the same run. Additionally, it facilitates incorporating new data engines, since the functionality to interoperate with a given engine is encapsulated in a wrapper that adopts a standardized interface.

We now present the experiments that we carried out with the workloads described in Section 2.2 and limiting the TPCx-BB query set to the SQL queries

only. For each run, we produce a chart that shows for the individual queries in the Elasticity Test, which of them satisfied their corresponding SLA and which ones violated it. In turn, the SLAs were defined by averaging the running times of a given query at the Power Test (thus, in isolation) for each system and adding a 25% percent margin. We present next results for the 1 TB and 10 TB scale factors.

For the 1 TB scale factor, we limit the number of query streams to 4, resulting in a total of 56 queries ($14x4$). The time interval between each batch of queries was set to 120 seconds. In relation to execution times and SLA satisfaction, Figure 5 shows the behavior of the different SUTs. Hive on ProvA reflects a relatively poor performance by a significant proportion of SLA violations. Concretely, the result for ProvA Hive is 71% SLA violations. The results are slightly worse for Hive on ProvB, for which 80% of the SLAs are violated. In both cases, the fact that execution times increase as the test progresses indicates that a significant backlog of queries is built as shown in Hive charts at 1TB in Figure 5.

In contrast, in the Spark results no SLAs were violated on any of the platforms. Furthermore, through the execution significant slack is left after the query is completed, as indicated by the gray areas in the charts. Similar execution times at the beginning and end of the test also indicate that a query backlog is not formed.

The largest data sizes considered in our experiments correspond to the 10 TB scale factor. Taking the smaller 1 TB scale factor as its precedent, the number of streams is now reduced from 4 to 2, resulting in a total of 28 ($14 * 2$) queries. The time between batches is doubled from 120 to 240 seconds as the cluster are not scaled with the increased data size. The workload generated for these experiments has the peculiarity that some batches have zero queries to be submitted, as depicted in Figure 6.

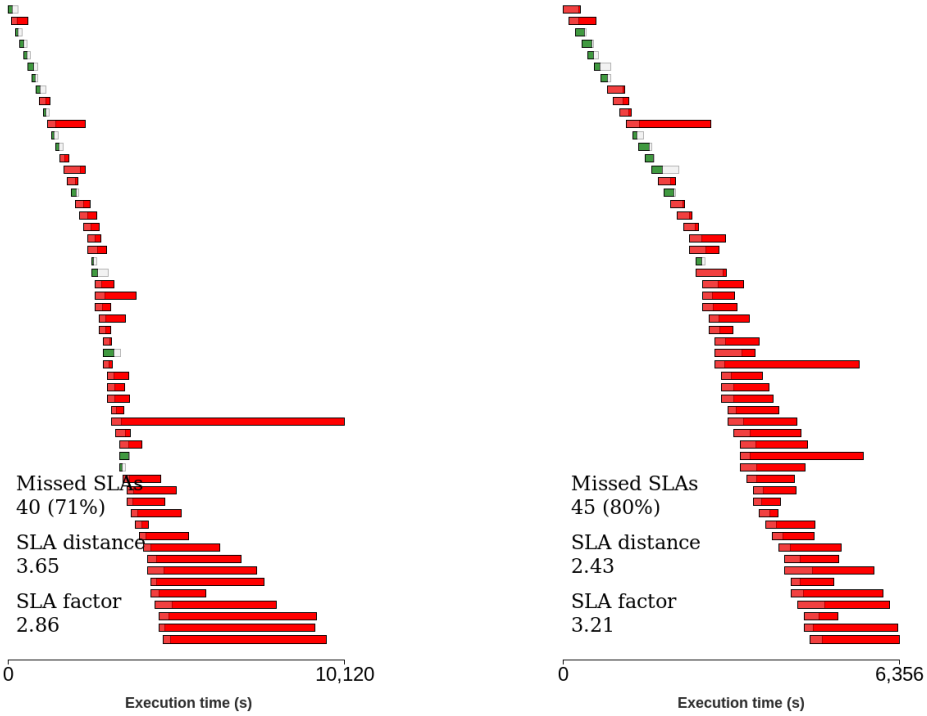
The experiments for only two of the SUTs considered completed successfully: Hive and Spark on ProvA; the result charts for these systems are presented in Figure 6, respectively. Spark violated the SLAs of only 3 queries, which represent 11% of the total in this experiment. Furthermore, these violations were not severe in terms of the additional time required to complete the queries, consequently, the SLA distance value for the run is low at 0.04. Hive missed the SLAs for 39% of the queries, 11 out of the 28 queries, to be exact. The SLA distance is 0.28 for Hive, due to many queries requiring significant time to complete.

4.3 Power test

We now present the summarized results of the TPCx-BB Power Test for the 1 TB, and 10 TB scale factors. Recall that the Power Test involves the consecutive execution of each of the queries in the specified list, in their sequential order. Thus, there is no concurrency, making this test useful to determine how efficient a system is to evaluate a particular query.

Figure 7 (left) presents the results for the 1 TB scale factor. A noteworthy observation is that the performance of Spark surpasses that of Hive for ProvB and ProvC, while ProvA exhibits the opposite result. The best performance corresponds to Spark on ProvB, while Hive on ProvC shows the poorest performance.

The results for the 10 TB scale factor are illustrated in Figure 7 (right). The best results correspond again to Spark but his time for ProvA. Hive on ProvA obtains



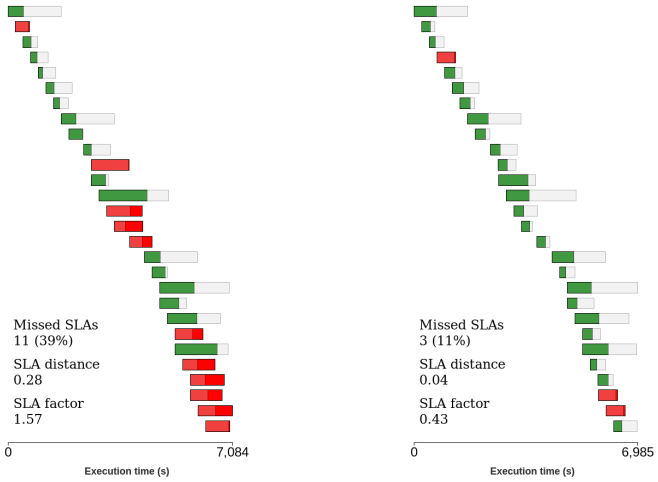
(a) ProvA Hive 1 TB

(b) ProvB Hive 1TB

(c) ProvA Spark 1 TB

(d) ProvB Spark 1 TB

Fig. 5: Elasticity Test results at 1 TB of Hive and Spark of 56 total queries. Queries in red have missed the SLA.



(a) ProvA Hive 10 TB (b) ProvA Spark 10 TB

Fig. 6: Elasticity Test results at 10 TB for ProvA Hive and Spark of 28 total queries.

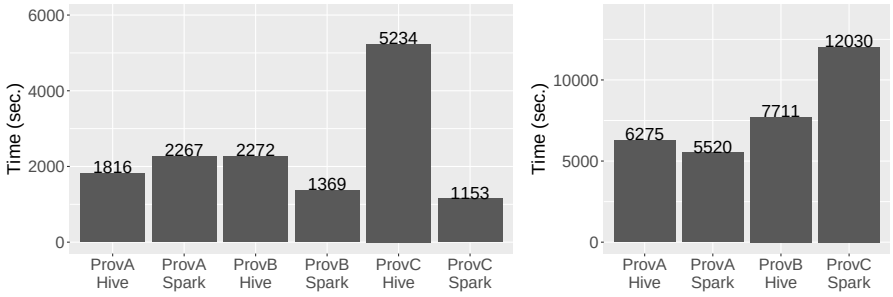


Fig. 7: Power Test times for the 1 TB (left) and 10 TB (right) scale factors.

the second-best results. Hive on ProvB shows a slightly poorer performance than Hive on ProvA. Finally, Spark on ProvC shows the poorest performance.

4.4 Throughput Test

We next present the results of the TPCx-BB Throughput Test. This test consists of issuing queries through a series of streams. The queries of a particular stream have to be completed serially, and according to an order established in the benchmark. Thus, in contrast with the Elasticity Test, the number of streams bounds the overall concurrency. Again, the 1TB and 10TB scale factors were used in these experiments, which were also limited to the 14 SQL queries. In the following charts, the times given correspond to the metric discussed in Section 3. Essentially, this metric is calculated by taking the time to complete the Throughput Test and dividing it by the number of streams.

The results for the 1 TB scale factor Throughput Test are presented in Figure 8 (left). The test used four query streams at this scale factor. Spark showed

adequate performance, clearly surpassing Hive for all providers. Particularly, the results Spark obtains with ProvB and ProvC represent the best among the SUTs; Spark with ProvA shows a completion time that is not much larger.

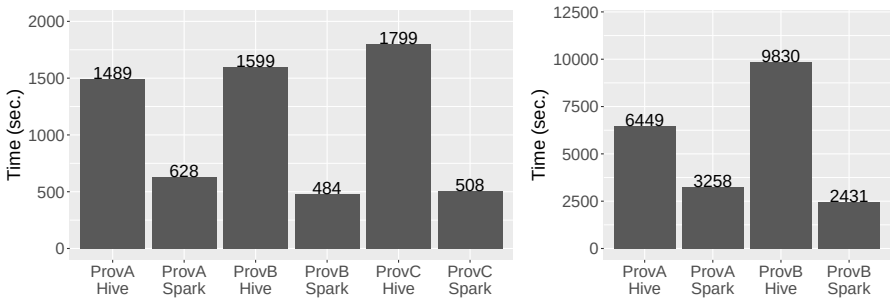


Fig. 8: Throughput Test times for the 1 TB (left) and 10 TB (right) scale factors.

We present the results for the 10 TB scale factor in Figure 8 (right), which consisted of two query streams. Spark shows again the best performance, with ProvB beating ProvA. In the case of Hive, the system relying on ProvA shows a better performance than that on ProvB, but both lag behind the Spark systems.

Based on the results of these tests, in the next section, we derive full benchmark metrics and also examine the corresponding costs.

5 Results analysis

In this section, we present the final performance scores for each system. We compare the result of the original TPCx-BB metric with the score obtained using the Elasticity Test introduced in Section 3. As in the TPCx-BB specification, we also consider the Price/Performance Metric, which is obtained by dividing the total cost of executing the benchmark by the performance metric. However, we present results based only of relative costs, since as stated earlier the providers are not identified and additionally, costs have exhibited a downward trend that makes the exact amounts less relevant.

5.1 BBQ++ metric

In Figure 9 (left) we present the results obtained using our new BB++Qpm metric at 1 TB, alongside with the original BBQpm score. We emphasize that the two values are not directly comparable and that the purpose of presenting them side by side is to see how the differences between systems and potentially the rankings change as a result of introducing the Elasticity Test. We can see that Spark on all providers achieves significantly higher scores than Hive at this scale factor, once elasticity is considered. Furthermore, for Hive we witness an enormous drop in performance for the new metric.

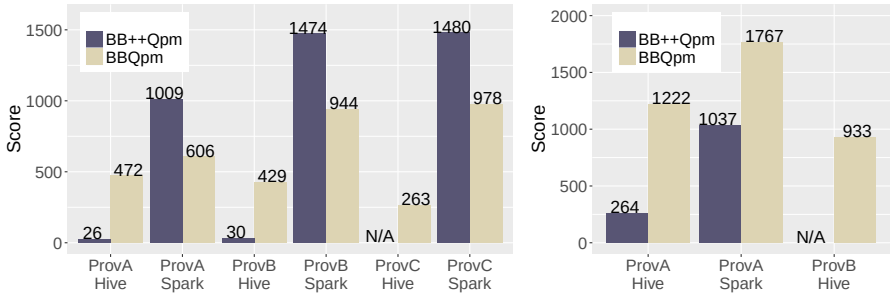
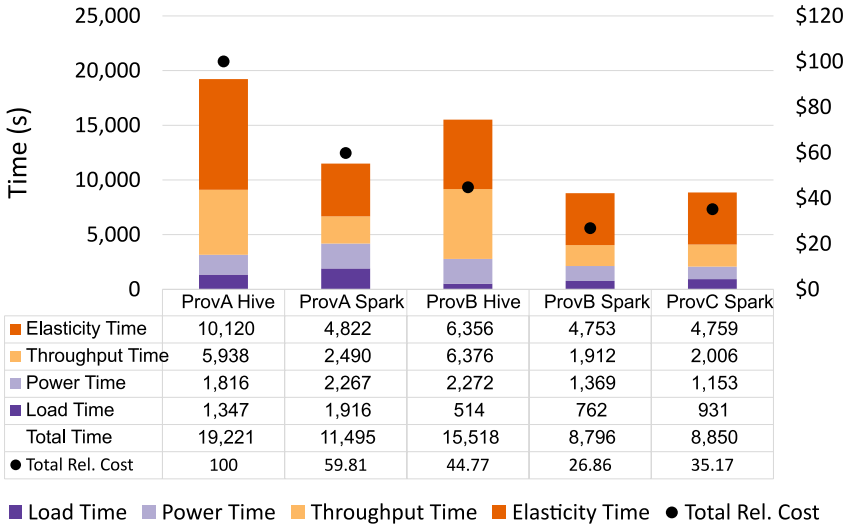


Fig. 9: BB++Qpm and BBQpm scores for the 1 TB (left) and 10 TB (right) scale factors.

In Figure 9 (right) we show the results at 10 TB. Hive struggled to meet the SLAs at this data scale. On the other hand, Spark on ProvA obtained a much better score because of its score in the Elasticity Test.

5.2 Costs



	ProvA Hive	ProvA Spark	ProvB Hive	ProvB Spark	ProvC Spark
Elasticity Time	10,120	4,822	6,356	4,753	4,759
Throughput Time	5,938	2,490	6,376	1,912	2,006
Power Time	1,816	2,267	2,272	1,369	1,153
Load Time	1,347	1,916	514	762	931
Total Time	19,221	11,495	15,518	8,796	8,850
Total Rel. Cost	100	59.81	44.77	26.86	35.17

Fig. 10: Total BigBench times and relative costs at 1 TB.

For our cost analysis, the benchmark execution is broken down into separate parts, the times of the individual tests of the benchmark are then added up to obtain the total time, then a total cost is calculated as a function of the total time. The cost calculation depends on the characteristics of each system. Since, as stated

earlier, we present only relative costs, a cost of $100 represents the maximum actual cost among the SUTs.

At the 1 TB scale factor, we obtain the costs presented in Figure 10. The costs of Spark are lower than those of Hive, with the ProvB and ProvC systems having the lowest costs overall. We also observe that the Elasticity Test plays the most significant role in determining the total execution time for most systems, with Hive on ProvB being the exception.

At the 10 TB scale factor, we were able to generate complete results only for Hive and Spark on ProvA. Due to space limitations, we omit a full time and cost breakdown and report only the total relative costs. For Hive the total relative cost was (as defined) $100, whereas it was $79 for Spark. Thus Spark turned out to be about 21% cheaper than Hive.

5.3 Price/performance score

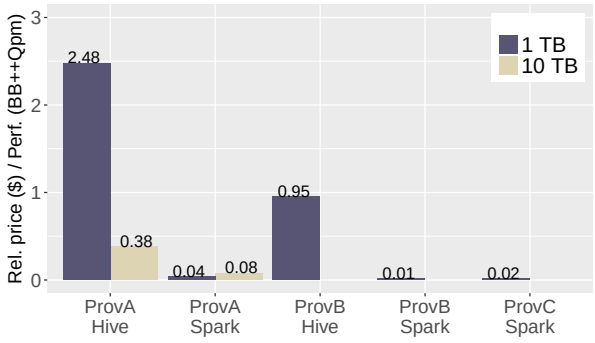


Fig. 11: Relative price/performance.

In Figure 11, we show the relative price/performance, computed as the total relative cost of the benchmark execution in dollars, divided by the BB++Qpm performance metric. Where available, results are shown at each of the scale factors that were tested. On this chart, lower is better.

We observe that there is a significant range in the price/performance at 1 TB. There is a large difference between Hive (worse) and Spark (better) on ProvA. Overall at 1 TB, Spark dominates, achieving very low price/performance on all providers. ProvA with Spark provides the best price/performance at 10 TB, the difference with respect to Hive on ProvA is significant but not as large as for the 1 TB scale factor.

6 Related work

A methodology to measure the elasticity of cloud services under varying workloads is presented in [2], we adopt similar techniques but also integrate our approach into

a full benchmark. SLAs have been applied in many contexts. For instance, their specification (through step-wise functions) and associated optimization problems are addressed for Service Oriented Architectures in [5]. Those and other alternative methods to define SLAs could be incorporated into our approach. Similar to our work, TPC-DS V2 is presented in [3], where the authors extend TPC-DS to consider data integration and multiple user aspects.

7 Conclusions

The Elasticity Test takes into consideration dynamic workloads and QoS aspects that are crucial in cloud environments i.e., fluctuating query submissions and multiple scale factors. We evaluate this extension by experiments on the cloud platforms of three major vendors and at two different scale factors, using two popular open-source big data processing systems. The analysis of the results of these experiments yields information of practical interest and validates our approach.

In particular, experiments show how certain configurations of the systems and platforms fail to meet SLAs under concurrency, scoring differently than the current TPCx-BB metric and reflecting poor resource isolation or scheduling. In contrast to the Throughput Test, which waits until a query finishes per concurrent stream to issue the next, our Elasticity Test summits queries following a more realistic pattern. This fact creates either high-intensity periods or queues, as well as low-intensity or even no new query arrivals. In this way testing more system components i.e., the scheduler and workload manager. Modern elastic services such of database or query as-a-service (DBaaS or QaaS) which can scale up a down compute nodes can benefit by saving in costs in low insensitive periods while adding more resources to prevent missing query SLAs. We believe an extension such as the Elasticity Test can complement current data TPC benchmarks to evaluate cloud analytical services.

References

1. Ghazal, A., Rabl, T., Hu, M., Raab, F., Poess, M., Crolotte, A., Jacobsen, H.A.: 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. SIGMOD '13, ACM, New York, NY, USA (2013)
2. Islam, S., Lee, K., Fekete, A., Liu, A.: How a consumer can measure elasticity for cloud platforms. In: Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering. pp. 85–96. ICPE '12, ACM, New York, NY, USA (2012)
3. Poess, M., Rabl, T., Jacobsen, H.A.: Analysis of tpc-ds: The first standard benchmark for sql-based big data systems. In: Proceedings of the 2017 Symposium on Cloud Computing. pp. 573–585. SoCC '17, ACM, New York, NY, USA (2017)
4. Ramakrishnan, R.e.a.: Azure data lake store: A hyperscale distributed file service for big data analytics. In: Proceedings of the 2017 ACM International Conference on Management of Data. pp. 51–63. SIGMOD '17, ACM, New York, NY, USA (2017). https://doi.org/10.1145/3035918.3056100, http://doi.acm.org/10.1145/3035918.3056100
5. Zhang, L., Ardagna, D.: Sla based profit optimization in autonomic computing systems. In: Proceedings of the 2Nd International Conference on Service Oriented Computing. pp. 173–182. ICSOC '04, ACM, New York, NY, USA (2004)