When Free Is Not Really Free: What Does It Cost to Run a Database Workload in the Cloud?

Avrilia Floratou[1], Jignesh M. Patel[1], Willis Lang[1], and Alan Halverson[2]

[1] University of Wisconsin-Madison, U.S.A.
{floratou,jignesh,wlang}@cs.wisc.edu
[2] Microsoft Jim Gray Systems Lab, U.S.A.
alanhal@microsoft.com

Abstract. The current computing trend towards cloud-based Database-as-a-Service (DaaS) as an alternative to traditional on-site relational database management systems (RDBMSs) has largely been driven by the perceived simplicity and cost-effectiveness of migrating to a DaaS. However, customers that are attracted to these DaaS alternatives may find that the range of different services and pricing options available to them add an unexpected level of complexity to their decision making. Cloud service pricing models are typically 'pay-as-you-go' in which the customer is charged based on resource usage such as CPU and memory utilization. Thus, customers considering different DaaS options must take into account how the performance and efficiency of the DaaS will ultimately impact their monthly bill. In this paper, we show that the current DaaS model can produce unpleasant surprises – for example, the case study that we present in this paper illustrates a scenario in which a DaaS service powered by a DBMS that has a lower hourly rate actually costs more to the end user than a DaaS service that is powered by another DBMS that charges a higher hourly rate. Thus, what we need is a method for the end-user to get an accurate estimate of the true costs that will be incurred without worrying about the nuances of how the DaaS operates. One potential solution to this problem is for DaaS providers to offer a new service called Benchmark as a Service (BaaS) where in the user provides the parameters of their workload and SLA requirements, and get a price quote.

1 Introduction

One of the greatest hurdles associated with deploying traditional on-site relational database management systems (RDBMSs) is the overall complexity of choosing, configuring, and maintaining the RDBMS as well as the server it operates on. In choosing and configuring a particular RDBMS and server to deploy, the users must have a firm understanding of the characteristics of their particular workload. Some of the important characteristics include the size of the database, the nature of the queries (transactional or ad-hoc/analytic), and the desired metric of performance (latency or throughput). Along with the upfront decisions of a particular RDBMS and corresponding server, the user must consider the long-term licensing, maintenance, and administration costs of running

R. Nambiar and M. Poess (Eds.): TPCTC 2011, LNCS 7144, pp. 163–179, 2012.

the system. This complexity that is associated with managing onsite DBMSs is a key reason why cloud-based Database-as-a-Service (DaaS) is starting to gain in popularity as an alternative to on-site RDBMS systems, especially for small and mid-sized database users.

The widely perceived advantage of the DaaS paradigm is that the user has now transferred the complex and nuanced decisions, and the heavy costs of operating an on-site RDBMS to the DaaS provider. Specifically, by turning to a DaaS, the user stores the data in the DaaS, and uses the DaaS APIs to query their data, for a monthly subscriber fee. This monthly fee incorporates all the responsibilities (such as data availability) that the provider has taken on. This fee also includes an "on-demand" payment model for computing resources that are consumed (this later component includes the costs that are associated with the CPU cycles and the storage that is consumed). However, the DaaS providers recognize that the needs of the database users varies significantly, and that one fixed pricing model will alienate one or more segments of the customer market. Consequently, in order to appeal to the entire spectrum of potential users, the DaaS providers have begun to diversify their offerings with multiple pricing options, each promising different levels of computing power, storage capability, and measures of performance. However, from the users' perspective, there is now a bewildering set of choices. As with the process of choosing an on-site RDBMS, they must now fully understand the characteristics, such the raw DBMS performance and query workload characteristics, when choosing an appropriate DaaS product. In fact, with the addition of the pay-as-you-go model for the computing resources, they now have an additional factor to consider – namely, the impact of the computing resources usage on their bottom line.

Initially, it may seem that the DaaS products alleviate many of the pains that are associated with running an on-site RDBMS. However, as we show in this study, the truth is that the users are actually in a tough position – they must now make an upfront decision of choosing a DaaS offering, while the long-term performance and cost consequences of their decisions are harder to figure out.

A crucial point that we make in this paper is that currently the DaaS users do not have an effective method to compare the suitability of one DaaS option over another, and fully understand the actual "cost" of their service. In a traditional RDBMS setting, the database users know that they can always turn to well-established benchmarks (such as the TPC benchmarks), to estimate whether one solution is more suitable than another. However, while such benchmarks identify price and performance as key metrics, these metrics have not been defined for the complex variable pricing models of DaaS products. For instance, they do not consider storage costs of the database or the utilization hours as factors of the price/performance. Moreover, TPC benchmarks usually take into consideration the total cost of ownership as a primary metric. This is incompatible with the "pay-as-you-go" model of cloud computing since the cloud customers are not directly exposed to the hardware, software maintenance, and administration costs of the deployment.

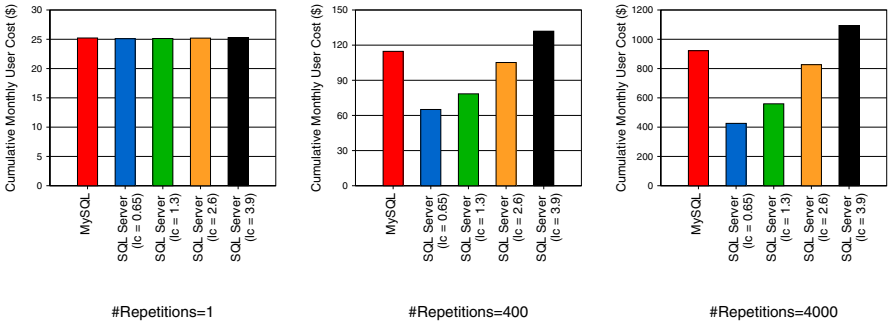

Fig. 1. Cumulative monthly user cost as a function of workload repetitions, DBMS type, and pricing model

To highlight the practical need for an easy to use and accurate pricing model, consider the popular Amazon Relational Database Service (RDS) [1]. While Amazon initially provided users with a database service backed by MySQL [11], recently they have unveiled an option to swap the back-end to an Oracle RDBMS instance [12]. Of course, these two options are not price equivalent, and currently the "Quadruple Extra Large DB" instance cost of the MySQL option is $2.60 per compute-hour, while the Oracle option is 31% more expensive at $3.40 per compute-hour. This price difference is largely due to the licensing cost ($0.80 per compute-hour) of the commercial Oracle system over the open-source MySQL system. While a cursory glance at these numbers would suggest that the cost-conscious user should buy the MySQL option, this choice ignores the fact that the often superior performance of a commercial DBMS may actually result in less computation time than the "free" MySQL option, and thus may actually be the cheaper option in some cases!

To better illustrate this point, consider running the following Wisconsin benchmark [7] query (Query 21):

```
INSERT INTO TMP
SELECT MIN (unique3) FROM TABLE1
GROUP BY onePercent
```

When we run this query on MySQL and a commercial DBMS (SQL Server) on the same physical machine (configuration details are described in Section 3), SQL Server runs this query in 185 seconds while MySQL takes 621 seconds to execute this query. How is the user's cost affected by this 3.3X performance gap when the user decides to run this workload on a DaaS?

Assuming a simple pricing model where the user pays a fixed cost of $1.30 per compute-hour for the specific DB Instance Class used, and a monthly storage fee of $25 for a database of 250GB, Figure 1 shows the cumulative monthly cost for the full deployment when these two RDBMSs are used, and when the workload consists of repetitions of the above query. For the SQL Server-based service, the user has to pay an extra hourly license fee/cost. Figure 1 examines four possible pricing models for the hourly license costs (`lc`) ranging from $0.65 to $3.90.

Interestingly, Figure 1 shows that although the user does not need to pay any license fee for MySQL, using this DBMS results in a higher total cost when the license fee for SQL Server is less than $3.90, and the user frequently issues such query requests. If the user load is high (e.g., around 4000 queries/month), then choosing the commercial-based RDBMS can save the user up to 54%. The reason for this behavior is that the higher efficiency of SQL Server results in decreased resource usage, and overall reduces the end-to-end cost for the user.

1.1 Towards BaaS

As the example above illustrates, the actual cost that a DaaS user will incur is hard to guess upfront. A simple approach to solve this pain point is to take the existing pricing model a step further. So in this scenario, the user provides the database and workload characteristics to the DaaS provider and in return the DaaS provider gives a price quote for running that workload. Along with the price quote the DaaS provider can attach a Service Level Agreement (SLA) that make guarantees on aspects such as performance variability and availability, and also lays out the penalty associated with SLA violations. (Alternatively, some parameters in the SLA could be provided upfront by the user, or the DaaS provider may come back with multiple price quotes at different SLA levels.)

Thus, what we need is for the DaaS providers to run another service – namely, "*Benchmark as a Service*" (BaaS) that makes it transparent to the user what it would cost to run their workload. Such a BaaS service could be free, and then would be crudely analogous to the utility model that is present in other parts of our lives – for example, internet service providers give an accurate price and specify the upper limit of the bandwidth. We acknowledge that a DaaS provider may have a more complicated problem at hand since the SLAs in a DaaS setting could be complex (hence, this is a promising direction for future work). But, from the perspective of the end-consumer of DaaS, a transparent pricing model could be very appealing, and perhaps a competitive advantage for the DaaS providers that choose to simplify their DaaS offering by coupling it with a BaaS service.

The BaaS approach also has a number of potential advantages for the DaaS provider as it provides a strong motivation to find the most optimal way of running the backend DBMS engine (rather than punting this decision to the end user), thereby reducing their operational cost (and perhaps improving their bottom line). Furthermore, the BaaS approach may provide more flexibility in managing the DaaS infrastructure – for example, a DaaS provider may not need to offer a range of DBMSs or data processing backends, and could simplify their infrastructure management by using only a single data processing engine. Finally, with a BaaS approach, the overall DaaS system potentially operates at a much higher operating efficiency (generally the queries across the system are likely to run far more efficiently then when the end user has to make nuanced decisions about configuring their DBMS and making bad choices), which in most cases is also likely to produce a more energy-efficient way of operating the DaaS, since in many cases the goal of energy efficiency lines ups with the goal of optimizing for traditional performance goals.

The remainder of this paper is organized as follows: Section 2 presents our cost model. Experimental results are presented in Section 3, while Section 4 discusses related work. Section 5 contains our concluding remarks and points to some directions for future work.

2 Cost Model

This section presents a simple cost model for using relational DBMSs in the cloud. The model considers the cost that is incurred by the end user in using a DaaS offering. We then use this model in our experiments (see Section 3) to explore the costs associated with using a DaaS product.

We patterned a simple pricing model crudely using Amazon's DaaS product as a reference. According to the Amazon's DaaS pricing model [1], the users pay only for the resources that they consume. Several parameters determine this cost. The first one, is an hourly fee that corresponds to the specific DB Instance Class chosen by the customer. The DB Instance is a database environment in the cloud with the compute and storage resources that the customer specifies. For example, currently in Amazon's RDS, 6 DB Instance Classes are provided. The "Small" DB Instance Class has 1.7GB of main memory and one 1.0-1.2 GHz CPU core (1 ECU), whereas the "Extra Large DB Instance" has 15GB of main memory and four 2.0-2.4 GHz CPU cores (8 ECUs). Generally, the hourly rates vary with the DB Instance Classes, since each class has different hardware characteristics. An extra hourly license cost/fee, is added for DB Instances backed by a commercial DBMS, which also varies according to the DB Instance Class chosen. The last parameter is a monthly storage fee per GB of the provisioned storage needed by the workload.

Consider a fixed database instance type chosen by the user with corresponding hourly cost `dbc`, an hourly license fee for the DBMS equal to `lc`, a monthly fee for the provisioned storage per GB equal to `stc`, and `H` hours of utilization per month of the DB instance. Given that the DB instance has associated capacity of `DS` GB, the monthly user cost (MUC) can be determined as:

$$MUC = H * (dbc + lc) + DS * stc \tag{1}$$

To keep our model simple, we do not consider the monthly network related costs.We also do not consider the costs for the extra backup storage that may be needed. These rates affect the total cost in a way similar to the storage fee `stc` and can easily be added to the above equation. Moreover, our model assumes that only one database instance is used by the customer. Creating and validating a more complex model that considers a combination of different database instance classes and multiple database instances per class is part of future work.

3 Experimental Evaluation

In this section, we discuss our experimental results which include performance measurements of a database server running different workloads and using different storage organizations, using MySQL and SQL Server. Based on these

performance results and the pricing model presented in Section 2, we compare the total cost that the user has to pay when using these two DBMSs in a DaaS.

The work by Schad et al. [13] presents experimental results showing that performance unpredictability is a major issue when running workloads in the cloud. The variance observed can be attributed to several factors, including different types of virtual systems provided by the service, different availability zones (distinct locations that are insulated from failures in other availability zones), and time of the day/week when the workload was run. Similar observations are discussed in the work of Armbrust et al. [2]. All these parameters make it difficult to estimate the impact on the cost and the performance of different database systems serving applications in a cloud-based environment. In this study, in an effort to eliminate these variances, we decided to measure the performance of the different DBMSs on a stand-alone local server machine. We show that even in this isolated environment, where variance due to the factors mentioned above is eliminated, the impact of the workload type and the efficiency of the DBMS on the monthly user's bill is not straightforward to estimate.

3.1 Server Configuration

Our test platform is a HP Proliant server with a dual quad-core hyperthreaded Intel Xeon L5630 processors (@ 2.13GHz), 32 GB of memory, and 12 HP 146GB 10K RPM SAS drives.

The server is dual booted with 64-bit Ubuntu Server 9.10 and 64-bit Windows Server 2008 R2 Enterprise Edition. The Linux version is used to run MySQL (MySQL Community Server 5.5.9) and the Windows version to run SQL Server 2008 R2 (Data Center Edition). Each disk is partitioned roughly evenly between the two operating systems. The first hard disk is used for the installation of the operating systems and all the database binaries.

3.2 DBMS Configuration

In our experiments, the database buffer pool is set to 24GB for both DBMSs. One disk is used to store the log files and the remaining 10 disks are reserved for the data files and the temporary space that is needed during query execution.

For SQL Server, we created a "file group" of 20 data files across the 20 data disk partitions (the 10 Windows partitions are further subdivided into two partitions). In this way, each of the 16 (hyperthreaded) cores can be assigned to one disk partition to allow parallel query processing. MySQL currently does not support such intra-query parallelism. For this reason, we created one data file striped across the 10 data disks so that we can get a high aggregate disk bandwidth. For MySQL we used the InnoDB storage engine, which is the default setting and the one used in Amazon's RDS.

3.3 The Wisconsin Benchmark

For our experiments we decided to use workloads based on the Wisconsin benchmark [7]. Our decision was driven by the fact that it is a simple "mi-

cro" benchmark that is fairly easy to set up and does not have complicated rules about how to run and measure a benchmark. Furthermore, this benchmark contains a variety of queries including selections, joins, projections, aggregations and updates. These simple queries are building blocks for more complex workloads and provide good insights about the potential impact on more complex workload characteristics.

The benchmark uses three basic relations, two that have the same number of tuples (T) and one that contains $T/10$ tuples. Each relation consists of sixteen attributes, thirteen 4-byte signed integers and three 52-byte varchars. The most widely used attributes in the benchmark are `unique1`, `unique2` and `onePercent`. The values of the `unique1` attribute are uniformly distributed unique random numbers in the range 0 to $T-1$. The values of `unique2` are in sequential order from 0 to $T-1$. The original benchmark paper [7] contains more information about each attribute and its values.

The benchmark explores two different kinds of storage organizations. The first one contains one heap file for each relation, and is called `StorageOrg-H`. This storage layout doesn't contain any primary key indices. In the second storage organization, called `StorageOrg-I`, each relation has a clustered index on the `unique2` attribute, a unique non-clustered index on the `unique1` attribute and a non-unique non-clustered index on the `onePercent` attribute.

3.4 Experimental Setting

For our experiments, we created six different types of workloads based on the Wisconsin benchmark. The first two workloads contain all the queries in the benchmark, and are called `MixedWorkload1` and `MixedWorkload2`. The first workload, `MixedWorkload1`, uses heapfiles as the storage layout (`StorageOrg-H`), whereas the second workload, `MixedWorkload2`, uses the clustered and non-clustered indices defined by the benchmark (`StorageOrg-I`). We generated a DSS-like workload using a subset of the Wisconsin benchmark queries. From this set of queries, we created two DSS workloads, `DSSWorkload1` and `DSSWorkload2`, corresponding to the two storage layouts (`StorageOrg-H` and `StorageOrg-I` respectively). Similarly, we generated two OLTP workloads consisting of OLTP-like queries. These two workloads are `OLTPWorkload1` and `OLTPWorkload2`, and correspond to the storage layouts `StorageOrg-H` and `StorageOrg-I` respectively.

Note that some of the queries of the mixed workloads are not presented in the OLTP or in the DSS workloads. More specifically, the 10% selection queries (Q2, Q4, Q6) as well as the 1% selection to screen query (Q8) are only included in the mixed workloads. We did not include the 10% selections in the other workloads because we wanted to experiment with high-selective queries in the OLTP workloads, and we wanted the DSS workloads to mainly consist of join and aggregation queries. Query Q8 was omitted since most of its execution time with MySQL was spent in printing the output to the screen, and not actually evaluating the query result. In the original Wisconsin benchmark paper [7], some of the queries are executed only on either `Storage-H` or `Storage-I`. In this work,

we decided to execute all the queries using both storage layouts. This decision was driven by the fact that for some queries the DBMSs don't pick the execution plan described in the benchmark. For example, Query 6 is supposed to use a non-clustered index, that's why it is tested only in `Storage-I`. However, in our experiments the actual plan picked by the optimizer of both DBMSs is a scan on the table. That's the reason why some of the queries are presented twice in some workloads (e.g., Q6 in `MixedWorkload1` and `MixedWorkload2`).

We created three data files using a Wisconsin benchmark generator. Each file corresponds to one relation of the benchmark. The two tables of the database contain 400M tuples, and the third one has 40M tuples. The size of the flat files for these tables is 80GB, 80GB, and 8GB respectively. Thus, the total raw database size is approximately 168GB. Between the executions of queries we purge the buffer pool (i.e., all reported numbers are "cold"). We also update the statistics for all the tables that are used in a query before its execution starts. The time to clean the buffer pool and update the statistics is not included in the experiment's total execution time. The temporary (TMP) tables that are used to store the results of each query are dropped after the query is executed and recreated when needed. Each query was run 3 times and the average value is reported. We did not see a lot of variance across runs of the same query. All the time values are reported in seconds. The data loading times were fairly similar across both DBMSs, and are not included in computing the total cost below.

We used the model presented in Section 2 to estimate the total cost incurred by the end DaaS subscriber/user. To compute the monthly user cost (MUC), we set the DB instance fee (`dbc`) to \$1.30 per hour. This `dbc` is equal to the rate of a high-memory double extra large DB Instance offered in Amazon RDS, which is the closest Amazon Instance configuration to our server. To get a better sense of how the total cost is affected by the license cost/fee (`lc`), we experimented with the following hourly license rates for the commercial DBMS: $\{\$0.65, \$1.30, \$2.60, \$3.90\}$. Since MySQL is open-source, its licensing fee is \$0. The monthly storage fee `stc` is set to \$0.10 per GB (similar to Amazon's RDS rate). We set the provisioned storage `DS` (data, log files and temporary space) for both DBMSs to 250GB.

To evaluate how the storage fee combined with the hourly fees affects the monthly user cost, we varied the number of repetitions of the workload, so that we can experiment with short and long-running workloads of the same type. We first report the cumulative user cost when the workload is executed only once (#repetitions=1). The next number of repetitions reported (#repetitions=N), corresponds to a total execution time close to a period of one month (computed based on the execution time of the workload on the slowest DBMS). This case represents the scenario were the end user application is driving the provisioned DBMS instance nearly to its peak capacity (for the slowest DBMS). Finally, we also present the comparative monthly costs when $N/10$ repetitions are performed. For example in Figure 1, $N = 4,000$, since the slowest DBMS (MySQL) can execute Query 21, approximately $4,000$ times in a period of a month.

Table 1. Mixed Workload 1

Query	Query Description	SQL Server Time (secs)	MySQL Time (secs)
Q1	1% selection on `unique2`	224	665
Q2	10% selection on `unique2`	482	1185
Q5	1% selection on `unique1`	195	739
Q6	10% selection on `unique1`	332	1191
Q7	Single tuple selection to screen	191	555
Q8	1% selection to screen	236	1721
Q18	1% projection	129	1523
Q20	Min. aggregate	190	482
Q21	Min. aggregate with group by	185	621
Q22	Sum aggregate with group by	187	747
Q26	Insert 1 tuple	0.20	0.23
Q27	Delete 1 tuple	192	637
Q28	Update on `unique2`	192	595
Q32	Update on `unique1`	197	609
Total		**2932**	**11270**

Table 2. Mixed Workload 2

Query	Query Description	SQL Server Time (secs)	MySQL Time (secs)
Q3	1% selection on `unique2`	22	51
Q4	10% selection on `unique2`	203	551
Q6	10% selection on `unique1`	883	1146
Q7	Single tuple selection to screen	0.75	0.60
Q8	1% selection to screen	62	1245
Q12	JoinAselB	412	1071
Q13	JoinABPrime	408	1004
Q14	JoinCselAselB	583	1512
Q18	1% projection	864	1495
Q23	Minimum aggregate	0.21	0.83
Q29	Insert 1 tuple	0.99	0.57
Q30	Delete 1 tuple	0.65	0.66
Q31	Update on `unique2`	1.47	0.73
Q32	Update on `unique1`	0.75	0.71
Total		**3441**	**8079**

3.5 Mixed Workloads

The mixed workloads contain all the queries in the Wisconsin benchmark that finished within 3 hours with both DBMSs. Some queries (i.e., MySQL running joins in `MixedWorkload1`) were stopped after 14 hours of execution. Although the same queries were completed using SQL Server, we do not take into account these numbers. It is clear that having such queries in the workload will lead to poor performance and higher cost, and hence will favor the usage of the commercial DBMS. However, we believe it's interesting to see what happens with respect to performance and cost when all the queries of the workload are completed in both systems in a reasonable amount of time. Note that all the queries that MySQL could finish within 14 hours were also completed by SQL Server within 14 hours.

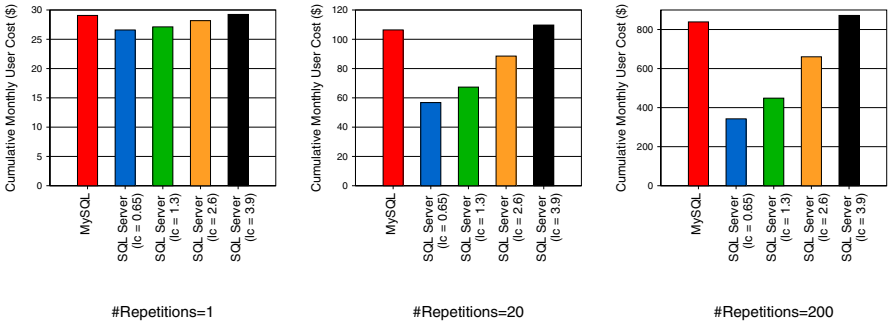

Fig. 2. Cumulative monthly user cost as a function of workload repetitions, DBMS type and pricing model (`MixedWorkload1`)

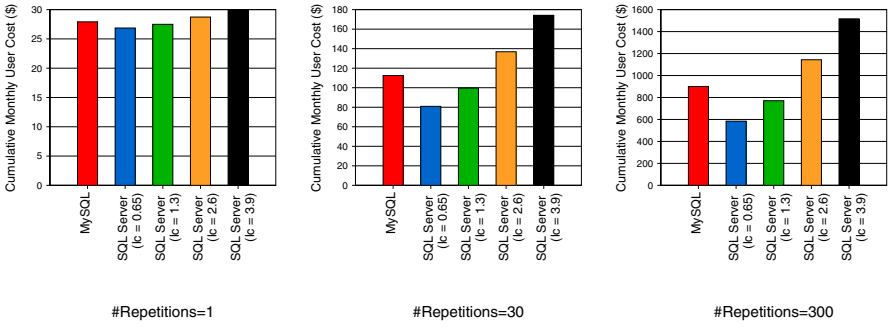

Fig. 3. Cumulative monthly user cost as a function of workload repetitions, DBMS type and pricing model (`MixedWorkload2`)

Tables 1 and 2 contain the execution times for both DBMSs using `MixedWorkload1` and `MixedWorkload2` respectively. The last rows of the tables contain the total execution time for each database system used. Figures 2 and 3 show the estimated monthly cost for the customer when MySQL or SQL Server is used.

As shown in Table 1, when the database consists only of heapfiles (`MixedWorkload1`), MySQL is approximately 3.84X (2.3 hours) slower than SQL Server. Notice that Table 1 does not show the original Wisconsin benchmark Queries 9-17 – these are join queries that did not complete with MySQL but completed using SQL Server in a reasonable amount of time (between 400-1000 seconds for each query).

Figure 2 shows how the total user cost is affected by the performance gap that exists between the two systems, when the repetitions of the workload as well as the hourly license fee for the commercial DBMS is varied. As shown in this figure, when the workload is executed only once, the difference in cost between SQL Server and MySQL is very small. In this case, the execution time is not long enough to make a significant impact, and thus the total cost is dominated by the monthly storage fee. The difference in the total cost between the two systems increases with the number of queries issued. As shown in the figure, the

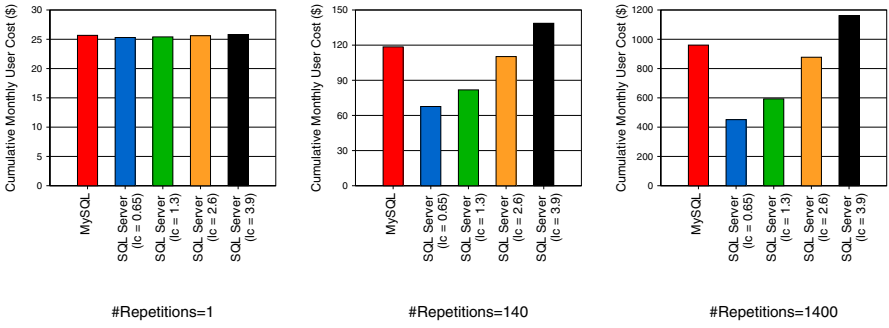

Fig. 4. Cumulative monthly user cost as a function of workload repetitions, DBMS type and pricing model (`DSS Workload 1`)

"free" open-source DBMS results in higher total cost when the license fee for the commercial DBMS is below $3.90. When the workload is executed 20 times the cost savings with SQL Server is 17%(lc = $2.60), 37% ($lc$ = $1.30) and 47% ($lc$ = $0.65). In the case of 200 repetitions (almost a month running time with MySQL), when the license fee is $2.60, using MySQL results in a 21% increase in the user's monthly bill. In the case of a license fee of $0.65, the increase is more significant(59%). Regarding, the performance of the `MixedWorkload2`, as shown in Table 2, when the clustered and non-clustered indices are used, MySQL is approximately 2.34X (1.28 hours) slower than SQL Server. In this case, the existence of the clustered index on the `unique2` attribute significantly improved the execution of some joins (Q12, Q13, Q14) as well selections (Q3, Q4) and updates (Q29, Q31). The existence of the non-clustered index on the `unique1` attribute improved the performance of the queries 30 and 32. However, it had an adverse impact on other queries (e.g Q5 in MySQL). This behavior can be attributed to the fact that the non-clustered index contains only two attributes: `unique1` and the primary key `unique2`. However, the query result contains all the 16 attributes of the relation. Evaluating this query using the non-clustered index as an access method possibly results in high random I/O behavior. A clustered index scan would probably result in a more efficient query execution (as was the case for the similar Q6 in both MySQL and SQL Server).

Figure 3 presents the total user cost similarly to Figure 2 for `MixedWorkload2`. As before, the free MySQL systems often results in higher costs, though now the license fee for the commercial DBMS has to be lower (around or below $1.30) than it was in Figure 2 to win over MySQL.

3.6 DSS Workloads

In this section, we evaluate the performance of the two DBMSs when the workload contains only decision-support queries. Similar to Section 3.5, based on these results and the cost model developed in Section 2 we estimate the total user cost for both cases. The DSS workload includes all the join and aggregation queries of the Wisconsin benchmark. Again, we report execution times only for the queries that were completed in both systems.

Table 3. DSS Workload 1

Query	Query Description	SQL Server Time (secs)	MySQL Time (secs)
Q20	Minimum aggregate	190	482
Q21	Minimum aggregate with 100 partitions	185	621
Q22	Sum aggregate with 100 partitions	187	747
All		**562**	**1850**

Table 4. DSS Workload 2

Query	Query Description	SQL Server Time (secs)	MySQL Time (secs)
Q12	JoinAselB	412	1071
Q13	JoinABprime	408	1004
Q14	JoinCselAselB	583	1512
Q23	Minimum aggregate	0.21	0.83
All		**1403**	**3588**

Regarding `DSSWorkload1`, as it is shown in Table 3, when using heapfiles as a storage layout only the aggregation queries were completed in both systems. In this case, MySQL was approximately 3.29X slower than SQL Server.

Figure 3 presents the total cost for the user varying the same parameters as in Figures 2 and 3. As it is shown in this figure, a similar pattern to that of `MixedWorkload1` is observed. The per hour cheap option (MySQL) does not always result in the lowest total cost. In fact, when the hourly license fee for SQL Server is less or equal to $2.60, choosing that over the free DBMS can result in cost savings of up to 53% (lc = $0.65, 1400 repetitions). On the other hand, using MySQL can result in cost savings of up to 17% when the license fee is equal to $3.90 and the workload is executed 1400 times.

Table 4 presents performance results for `DSSWorkload2`. The existence of the indices allows many joins to complete with MySQL, but negatively affected some aggregation queries. The reasons for this behavior are discussed in section 3.5. In this case, MySQL is 2.55X slower than SQL Server.

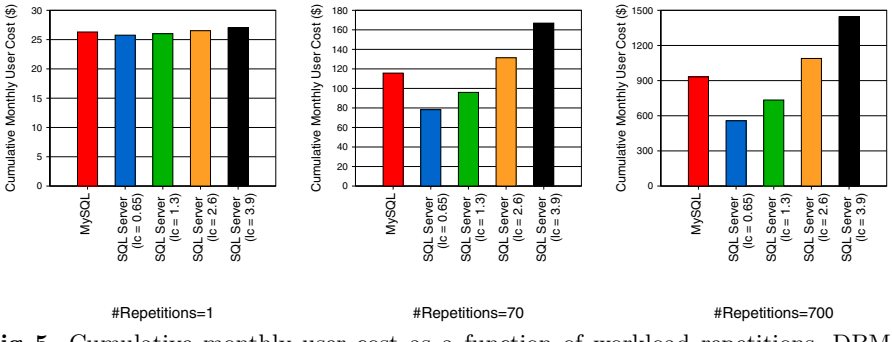

Fig. 5. Cumulative monthly user cost as a function of workload repetitions, DBMS type and pricing model (`DSS Workload 2`)

Table 5. OLTP Workload 1

Query	Query Description	SQL Server Time (secs)	MySQL Time (secs)
Q1	1% selection on `unique2`	224	665
Q5	1% selection on `unique1`	195	739
Q7	Single tuple selection to screen	191	555
Q26	Insert 1 tuple	0.2	0.23
Q27	Delete 1 tuple	192	637
Q28	Update on `unique2`	192	595
Q32	Update on `unique1`	197	609
All		**1191**	**3800**

Table 6. OLTP Workload 2

Query	Query Description	SQL Server Time (secs)	MySQL Time (secs)
Q3	1% selection on `unique2`	22	51
Q7	Single tuple selection to screen	0.75	0.60
Q29	Insert 1 tuple	0.99	0.57
Q30	Delete 1 tuple	0.65	0.66
Q31	Update on `unique2`	1.47	0.73
Q32	Update on `unique1`	0.75	0.71
All		**26.61**	**54.27**

The corresponding user cost is presented in Figure 5. Similarly to the previous results, the open-source DBMS is a more cost-effective choice when the license fee is greater or equal to $2.60. As before, the cost savings increases as the execution time increases, since in this case the monthly storage fee does not have a significant impact on the total cost.

3.7 OLTP Workloads

The OLTP workload consists of the queries of the Wisconsin benchmark that contain high-selective selections, insertions, deletions and updates. Similarly to the previous experiments, only the queries that were completed in both DBMSs are reported for each workload.

As shown in Table 5, when the database consists only of heapfiles, MySQL is 3.19X slower than SQL Server. The corresponding user's cost is presented in Figure 6. Similarly to the previous experiments, MySQL is the most cost-effective option when the hourly license fee is equal to $3.90. In all the other cases, the cost savings when using SQL Server can be as high as 51%.

When indices are used, MySQL is approximately 2X slower than SQL Server. Table 6 and Figure 7 present the performance results and the associated user cost.

3.8 Discussion

We have shown that the process of estimating the cost of a DaaS is not straightfoward, even in the simple case where the database system in not deployed in a virtualized environment and factors such as different availability zones, locations

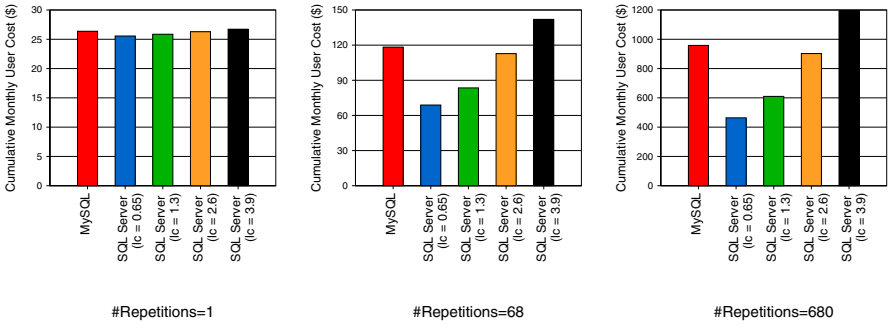

Fig. 6. Cumulative monthly user cost as a function of workload repetitions, DBMS type and pricing model (`OLTP Workload 1`)

or points of time are not taken into consideration. Parameters such as database efficiency, type of workload, and pricing model can all affect the resulting user cost. Consequently, often the option that initially seems cheap per hour, e.g., an open-source DBMS, can actually result in a higher monthly bill than that of a non-free, licensed DBMS.

4 Related Work

DBMS benchmarking is an age-old sport in the database community. The Wisconsin benchmark [7] was one of the first benchmarks developed for evaluating RDBMSs. Today, the series of the TPC benchmarks [16] are widely used for measuring the performance and the cost or relational database systems.

Following the advent of cloud computing, recent work has evaluated different cloud services on different types of workloads. More specifically, a recent paper [3] presents some initial ideas on what a general cloud benchmark should consider, focusing on the different kinds of cloud services and architectures and their corresponding pricing plans. One of the (many) considerations in this paper

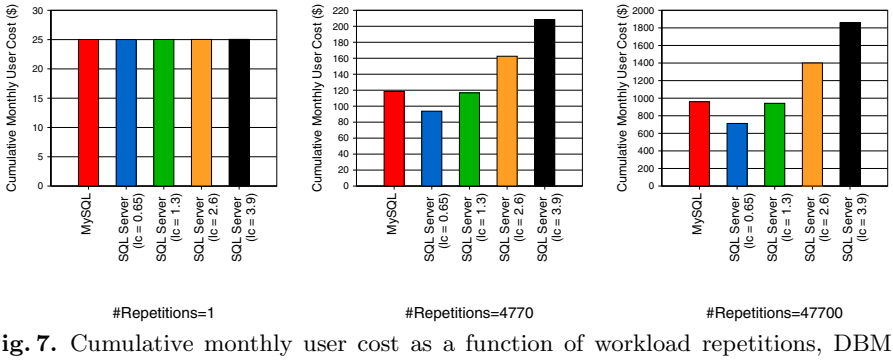

Fig. 7. Cumulative monthly user cost as a function of workload repetitions, DBMS type and pricing model (`OLTP Workload 2`)

is the end-user cost. A follow-up work [9] presents an evaluation of different cloud services when running enterprise web applications with OLTP workloads. Along the same lines, Berkeley's Cloudstone project [15] proposes a workload and metrics to study cloud infrastructures that deploy Web 2.0 applications. Comparing different cloud services has also been the focus of the recent work by Garfinkel [8], which evaluates three popular Amazon Computing Services (EC2, S3 and SQS). Another work [6] compares a traditional open-source RDMS and existing cloud computing technology (HBase). Cooper et al. [5] propose a benchmark to compare different popular datastores like Cassandra and PNUTS.

Virtualization techniques have been widely adopted in cloud-based environments. In a recent paper [10], the performance of relational database systems running on top of virtual machines has been studied. Bose et al. [4] present performance results from experiments running TPC database workloads on top of virtual machines, and make the case for a database benchmark on top of virtual machines. The follow-up work [14] presents a high-level overview of TPC-V, a benchmark designed for database workloads running in virtualized environments.

5 Concluding Remarks and Directions for Future Work

This paper has explored how two important dimensions in cloud environments, namely performance and cost, are influenced when different types of DBMSs are chosen by a DaaS user. More specifically, we have used a variety of simple workloads and storage organizations to evaluate two different relational DBMSs (one open-source and one commercial RDBMS). Our results show that given the range of the pricing models and the flexibility of the "on-demand" allocation of resources in cloud-based environments, it is hard for a user to figure out their actual monthly cost upfront. Interestingly, DaaS settings that at first sight seem cheaper per hour (since the backend is an open-source DBMS) and thus more-cost effective, can result in higher total costs in the long-run, since the backend DBMS may have poor performance characteristics on the users' workload. On the other hand, a DaaS setting backed by a high performance commercial DBMSs, while more expensive on a per hour basis, may be cheaper overall since its higher performance more than makes up for the hourly price differential. We note that these results should not be construed to mean that free open source DBMSs are always more expensive in the DaaS environment (or vice versa) – we have only tried two DBMSs in this paper, picking the most popular free open-source DBMS and a commercial DBMS. Rather, our work highlights that the real cost of running a workload in the DaaS is complicated, and may in some cases produce surprising results.

Thus, what we need is real transparency and clarity in pricing DaaS. An approach to this problem that we propose in this paper is "Benchmark as a Service" (BaaS), where by the DaaS provider can take the user workload as input (with SLA parameters) and provide an accurate price for that workload, or perhaps different prices at different SLA levels. This BaaS approach would move the DaaS offering closer to a true utility model (like gas and electricity, or internet service). But, we acknowledge that setting up a BaaS is challenging

as there are important aspects that need to be considered. For example, how to specify the workload. A starting point for describing this workload could be for the user to provide the database schema, average tuple sizes for each table, and a query set. But, additional parameters may be required, such as estimated database growth rates, or acceptable ranges for SLA parameters (e.g., query/workload response time or throughput). For simplicity from the users' perspective it is desirable that the workload specifications should not be overly complicated, but from the DaaS provider's perspective more details are probably required. Finding a good and practical balance is one direction for future work. Other aspects of future work include designing methods for a DaaS provider to efficiently run a mix of workloads that started with a BaaS, and monitoring and reacting to changes in workloads that started with a price quote from the BaaS.

Acknowledgement. This research was supported in part by a grant from the Microsoft Jim Gray Systems Lab, Madison, WI, and by the National Science Foundation under grant IIS-0963993. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding source or the organizations that employ the authors.

References

1. Amazon Relational Database Service, `http://aws.amazon.com/rds/`
2. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A., Zaharia, M.: Above the Clouds: A Berkeley View of Cloud Computing (2009)
3. Binnig, C., Kossmann, D., Kraska, T., Loesing, S.: How is the weather tomorrow?: towards a benchmark for the cloud. In: DBTest (2009)
4. Bose, S., Mishra, P., Sethuraman, P., Taheri, R.: Benchmarking Database Performance in a Virtual Environment. In: Nambiar, R., Poess, M. (eds.) TPCTC 2009. LNCS, vol. 5895, pp. 167–182. Springer, Heidelberg (2009)
5. Cooper, B.F., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R.: Benchmarking cloud serving systems with ycsb. In: SoCC, pp. 143–154 (2010)
6. Cryans, J.-D., April, A., Abran, A.: Criteria to Compare Cloud Computing with Current Database Technology. In: Dumke, R.R., Braungarten, R., Büren, G., Abran, A., Cuadrado-Gallego, J.J. (eds.) IWSM 2008. LNCS, vol. 5338, pp. 114–126. Springer, Heidelberg (2008)
7. DeWitt, D.J.: The wisconsin benchmark: Past, present, and future. In: Gray, J. (ed.) The Benchmark Handbook. Morgan Kaufmann (1993)
8. Garfinkel, S.L.: An evaluation of amazon's grid computing services: Ec2, s3 and sqs. Technical report (2007)
9. Kossmann, D., Kraska, T., Loesing, S.: An evaluation of alternative architectures for transaction processing in the cloud. In: SIGMOD Conference, pp. 579–590 (2010)
10. Minhas, U.F., Yadav, J., Aboulnaga, A., Salem, K.: Database systems on virtual machines: How much do you lose? In: ICDE Workshops, pp. 35–41 (2008)
11. MySQL, `http://www.mysql.com/`
12. Oracle Database, `http://www.oracle.com/us/products/database/index.html`

13. Schad, J., Dittrich, J., Quiané-Ruiz, J.-A.: Runtime measurements in the cloud: Observing, analyzing, and reducing variance. PVLDB 3(1), 460–471 (2010)
14. Sethuraman, P., Reza Taheri, H.: TPC-V: A Benchmark for Evaluating the Performance of Database Applications in Virtual Environments. In: Nambiar, R., Poess, M. (eds.) TPCTC 2010. LNCS, vol. 6417, pp. 121–135. Springer, Heidelberg (2011)
15. Sobel, W., Subramanyam, S., Sucharitakul, A., Nguyen, J., Wong, H., Klepchukov, A., Patil, S., Fox, O., Patterson, D.: Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0 (2008)
16. TPC Benchmarks, `http://www.tpc.org/information/benchmarks.asp`