# Benchmarking OLTP/Web Databases in the Cloud: The OLTP-Bench Framework

[Extended Abstract] *

Carlo Curino
Microsoft, USA
ccurino@microsoft.com

Djellel E. Difallah
U. of Fribourg, Switzerland
djelleleddine.difallah@unifr.ch

Andrew Pavlo
Brown University, USA
pavlo@cs.brown.edu

Philippe Cudre-Mauroux
U. of Fribourg, Switzerland
pcm@unifr.ch

## ABSTRACT

Benchmarking is a key activity in building and tuning data management systems, but the lack of reference workloads and a common platform makes it a time consuming and painful task. The need for such a tool is heightened with the advent of cloud computing—with its pay-per-use cost models, shared multi-tenant infrastructures, and lack of control on system configuration. Benchmarking is the only avenue for users to validate the quality of service they receive and to optimize their deployments for performance and resource utilization.

In this talk, we present our experience in building several ad-hoc benchmarking infrastructures for various research projects targeting several OLTP DBMSs, ranging from traditional relational databases, main-memory distributed systems, and cloud-based scalable architectures. We also discuss our struggle to build meaningful micro-benchmarks and gather workloads representative of real-world applications to stress-test our systems. This experience motivates the OLTP-Bench project, a "batteries-included" benchmarking infrastructure designed for and tested on several relational DBMSs and cloud-based database-as-a-service (DBaaS) offerings. OLTP-Bench is capable of controlling transaction rate, mixture, and workload skew dynamically during the execution of an experiment, thus allowing the user to simulate a multitude of practical scenarios that are typically hard to test (e.g., time-evolving access skew). Moreover, the infrastructure provides an easy way to monitor performance and resource consumption of the database under test. We also introduce the ten included workloads, derived from either synthetic micro benchmarks, popular benchmarks, and real world applications, and how they can be used to investigate various performance and resource-consumption characteristics of a data management system. We showcase the effectiveness of our benchmarking infrastructure and the usefulness of the workloads we selected by reporting sample results from hundreds of side-by-side comparisons on popular DBMSs and DBaaS offerings.

*Full version will appear at: http://oltpbenchmark.com.


*CloudDB'12,* October 29, 2012, Maui, Hawaii, USA.


## Categories and Subject Descriptors

H.2.4 [**Systems**]: Relational Databases—*benchmarking, performance, testing*; H.3.4 [**Systems and Software**]: Performance evaluation

## Keywords

Benchmarking, Testbed, Transactional, Web, OLTP, Workload

## 1. INTRODUCTION

Much of database and system research and development is centered around innovation in system architectures, algorithms, and paradigms, that deliver significant performance advantages or increase the hardware utilization efficiency for important classes of workloads. Therefore, in order to establish whether a piece of research or a development effort achieved its goals, it is often necessary to compare the performance or efficiency of alternative solutions targeting some representative workload. In this sense, benchmarking is central to most research and development efforts.

When considering data management systems, this implies: (1) the availability of representative workloads (data and data accesses), (2) an infrastructure to run such workloads against existing systems in a consistent and repeatable way (e.g., without introducing artificial bottlenecks), and (3) a way to collect detailed performance and resource-consumption statistics during the runs and compare them. At least in our experience in both academic and industrial settings, the three above components are often hard to come by. The net result is that significant effort is repeatedly devoted by independent researchers/practitioners in gathering or worse inventing workloads and in building *one-off* solutions for running such workloads and measuring performance. This is problematic for highly concurrent transactional or web workloads, since they are characterized by a multitude of small operations that must be executed in parallel by multiple clients and thus require a more sophisticated infrastructure and tighter performance monitoring.

Cloud platforms, with their pay-per-use billing models, shared multi-tenant infrastructure, and lack of control over infrastructure and tuning (e.g., Amazon RDS or SQL Azure), exacerbate the need for proper benchmarking solutions that are capable of accounting for performance and resources and executing representative workloads.

Each of the authors of this extended abstract has independently developed partial solutions for the issues listed above within previous research projects [1, 2]. This repeated and tedious work is what motivated us to join forces and try to provide a systematic,

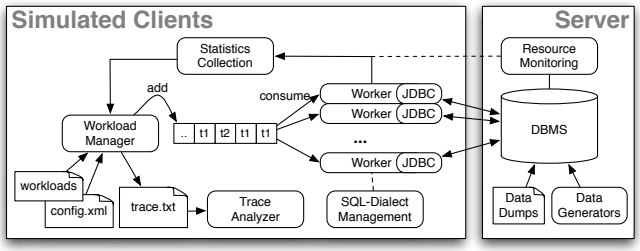



**Figure 1:** The architecture of the OLTP-Bench framework; On the left-hand side, the client-side driver handles workers and generates throttled workloads according to configuration provided by the user. On the right-hand side, the DBMS server monitor gathers resource utilization statistics.

reusable and extensible solution to this problem. In order to scope our effort, we decided to focus on transactional and web workloads targeting relational databases, although much of our design can be generalized beyond this scope. This was the seed of the OLTP-Bench effort, an open-source framework for benchmarking relational databases.

## 2. REQUIREMENTS AND ARCHITECTURE

From our experience and from extensive discussions with other researchers and practitioners we gather that, in order to simulate real-world scenarios and stress tests data management systems, a benchmarking infrastructure needs to provide:

- **(R1) Scalability**: the ability to drive the system under test at high transactional rates (i.e., test for max throughput);
- **(R2) Fine-Grained Rate Control:** the ability to control the rate of requests with great precision;
- **(R3) Mixed and Evolving Workloads:** the ability to support mixed workloads, and to change the rate, composition, and access distribution of the workloads dynamically over time;
- **(R4) Synthetic and Real Data & Workloads:** support for both synthetic and real data sets and workloads (and thus both programmatically-generated and trace-based executions);
- **(R5) Ease of Deployment/Portability:** the ability to deploy experimental settings and run experiments easily on a variety of DBMSs and DBaaSs using an integrated framework;
- **(R6) Extensibility:** the ability to extend the set of workloads and systems supported with a minimal engineering effort (e.g., centralizing SQL dialect localizations);
- **(R7) Lightweight, Fine-Grained Statistics Gathering:** the ability to collect detailed statistics of both client-side activity and server-side resource utilization with minimum impact on the overall performance;

Next we present the OLTP-Bench architecture (shown in Figure 1), and discuss how it fulfills the above requirements. OLTP-Bench is comprised of a Java-based client framework and a Python-based monitoring infrastructure (derived from DSTAT) on the server.

The user controls the behavior of the system by means of a configuration file, which instructs the system on how to connect to the DBMS under test, what experiment to run, the level of parallelism desired, and how to vary rate, and mixture over time. A second file is used optionally to replay an exact execution trace, and is leverage to support dynamic skew variations. Trace-based execution is useful for more faithful replay of real-world workloads, while programmatic generation of the load is ideal for synthetic workloads since it allows for more compact deployments (**R4**).

The client is organized with a centralized *Workload Manager* tightly controlling the characteristics of the load via a work queue. To achieve parallelism and high concurrency a user-specified number of worker threads consumes from the work queue and directly communicate with the system under test—in our current implementation via a JDBC interface.

The work queue offers a trade-off between requirements **R1**, **R2**, and **R3**; the centralized manager not only enables us to control the exact throughput rates and mixtures of the benchmark, but it also allows the workers to be light-weight enough to achieve good scalability—for YCSB we could push 12.5k transactions per second per CPU core over the network while tightly controlling speed and mixture of transactions. The currently released version is single-machine on the client side (while it supports a distributed DBMSs), but a multi-machine client version of OLTP-Bench is in the work.

Supporting multiple DMBSs is typically problematic, due to the many SQL dialects in existence. To cope with this we structured our codebase carefully so that SQL dialect "localization" is rather easy to achieve and thus adding support for new DBMSs or include a new workload running on all DBMSs we support is straightforward (**R5**, **R6**).

Workers collects detailed performance statistics, which are then aggregated and analyzed by the system which provides several common metrics such as throughput, and latency (at any percentile) with customizable windowing. The results are easy to correlate with the carefully time-aligned resource utilization statistics collected on the server side. OLTP-Bench supports single-node, DBaaS, and distributed DBMSs. Our discussion in this extended abstract focuses on the first two types, but we note that an earlier version of OLTP-Bench was used in our research on distributed systems [1, 2].

## 3. WORKLOADS

In this section, we briefly report about the set of workloads we implemented and that are released with the infrastructure.

Table 1 summarizes some of the statistics about the ten currently available workloads. While discussing each of the benchmarks is beyond the scope of this extended abstract, we would like to cite two examples to showcase the breadth of workloads included in our system:

**Resource Stresser:** This benchmark has been designed to provide a simple micro-benchmark for stress-testing specific hardware resources. As such the benchmark consists of transactions/queries that target individual resources (cpu, disk, locks, ram) and can be used to challenge a system in a specific area (e.g., how efficiently it handles high lock contention or disk-intensive workloads etc.). This benchmark is fully synthetic, and thus trivial to scale up and down and compact to ship.

**Wikipedia:** At the opposite side of the spectrum there is a benchmark derived from a long (and tedious) analysis of the actual Wikipedia website. We leveraged knowledge of the source code, the availability of the raw data, various interactions with the database administrators, and special access to extensive execution traces (10% of 3 months of the actual wikipedia website workload) to design a model of the actual wikipedia installation. Although we cannot claim this is an accurate model, much attention has been devoted to capture the key characteristics of the workload, and is thus a reasonable representation of the corresponding real-world application.

These two workloads exemplify different uses of our benchmarking infrastructure (e.g., performance debugging of a system, and testing against a real-world workload). The many other workloads we included complete this picture—more details on all of our workloads are available at **http://oltpbenchmark.com.**

One fundamental design principle of the OLTP-Bench project is that it does not impose any fixed set of configuration rules or predefined metrics of success for the benchmarks. We believe that

| Workloads | Tables | Columns | PKeys | Indexes | FKeys | # Txns | Read-Only Txns | # Joins | Application Domain | Operations |
|---|---|---|---|---|---|---|---|---|---|---|
| **AuctionMark** | 16 | 125 | 16 | 14 | 41 | 9 | 55.0% | 10 | On-line Auctions | Non-deterministic, heavy transactions |
| **Epinions** | 5 | 21 | 2 | 10 | 0 | 9 | 50.0% | 3 | Social Networking | Joins over many-to-many relationships |
| **JPAB** | 7[2] | 68[2] | 6[2] | 5[2] | 3[2] | 4[3] | 25.0% | N/A[3] | Object-Relational Mapping | Bursts of random reads, pointer chasing |
| **ResourceStresser** | 4 | 23 | 4 | 0 | 0 | 6 | 33.3% | 2 | Isolated Resource Stresser | CPU-, disk-, lock-heavy transactions |
| **SEATS** | 10 | 189 | 9 | 5 | 12 | 6 | 45.0% | 6 | On-line Airline Ticketing | Secondary indices queries, foreign-key joins |
| **TATP** | 4 | 51 | 4 | 5 | 3 | 7 | 40.0% | 1 | Caller Location App | Short, read-mostly non-conflicting trans. |
| **TPC-C** | 9 | 92 | 8 | 3 | 24 | 5 | 8.0% | 2 | Order Processing | Write-heavy concurrent transactions |
| **Twitter** | 5 | 18 | 5 | 4 | 0 | 5 | 0.9% | 0 | Social Networking | Client-side joins on graph data |
| **Wikipedia** | 12 | 122 | 12 | 40 | 0 | 5 | 92.2% | 2 | On-line Encyclopedia | Complex trans, large data, skew |
| **YCSB** | 1 | 11 | 1 | 0 | 0 | 6 | 50.0% | 0 | Scalable Key-value store | Key-value queries |

**Table 1:** Profile information for the benchmark workloads.

| Instance Type | CPU (v-core) | RAM | I/O Perf. |
|---|---|---|---|
| Small (S) | 1 EC2 (1) | 1.7G | Moderate |
| Large (L) | 4 EC2 (4) | 7.5 G | High |
| HighMem XLarge (XL-HM) | 6.5 EC2 (2) | 17.1 G | Moderate |
| HighMem 2XLarge (2XL-HM) | 13 EC2 (4) | 34.2 G | High |
| HighMem 4XLarge (4XL-HM) | 26 EC2 (8) | 68.4 G | High |

**Table 2:** EC2 RDS Experimental Systems

standardizations of how to run a benchmark are often biased by the proposers opinions and their context (e.g., the hardware available at the time). As evidence consider the many alterations of popular benchmarks that appear in various papers (it is rare to see a paper using a benchmark as it was originally intended). Therefore we concentrate our attention on providing a broad spectrum of raw functionalities and workloads and argue that, if this effort is successful, different user groups will spontaneously standardize its use in ways we are unlikely to foresee today.

# 4. SAMPLE EXPERIMENTS

We now present a few experiments sampled by the hundreds we ran. The experiments we report are designed to highlight the functionalities of our benchmarking infrastructure and not to compare or judge alternative DBMSs and DBaaS offerings.

The experiments we present have all been executed on Amazon's EC2 platform. Each of the DBMSs were deployed within the same geographical region, with a single instance dedicated to workers and collecting statistics and another instance running the DBMS server. For each DBMS server, we used a four virtual core instance with an 8GB buffer pool (this was sufficient to accommodate the working-set size of our workloads). The worker's transaction isolation levels were set to *serializable*. We flushed each system's buffers before each experiment trial. All of the changes made by each transaction were logged and flushed to disk by the DBMS at commit time. As reported in [3], EC2 is a "noisy" cloud environment where machine variability and cluster conditions can significantly affect benchmark results. To mitigate such problems, we ran all of our experiments with restarting our EC2 instances as little as possible, and executed the benchmarks multiple times and averaged the results.

## 4.1 Performance-vs.-Cost Comparison

First, we use OLTP-Bench ability to run at high throughput and to carefully monitor performance to measure the (max-throughput) performance-vs.-cost ratio of a single DBaaS provider using different workloads. Such a comparison allows a user to decide what the right trade-off is between performance and cost for their application. We pick YCSB and Wikipedia as candidate benchmarks for this test and leverage OLTP-Bench to test against five different instance sizes on Amazon's RDS (Table 2). We then ran each benchmark separately at its maximum speed for a total of 30 minutes. We show average maximum sustained throughput and 95th percentile latency from the middle 20 minutes of the experiment.

The graph in Fig. 2 shows throughput and latency measurements collected by OLTP-Bench compared to the different instance sizes. For YCSB, the L instance yields the best cost/performance ratio, with good overall throughput and low latency. Anything beyond

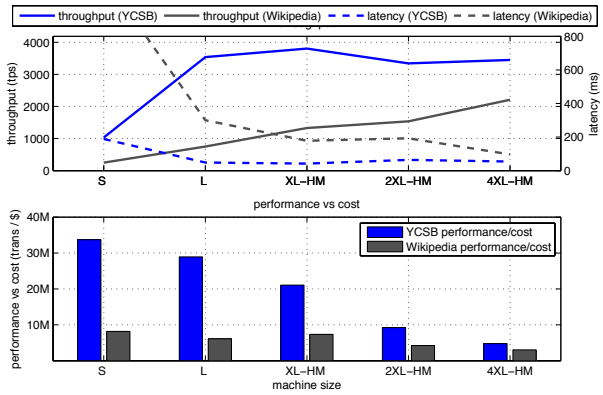



**Figure 2:** DBaaS Performance-vs-Cost – Comparing RDS using the Wikipedia and YCSB workloads within same data center.

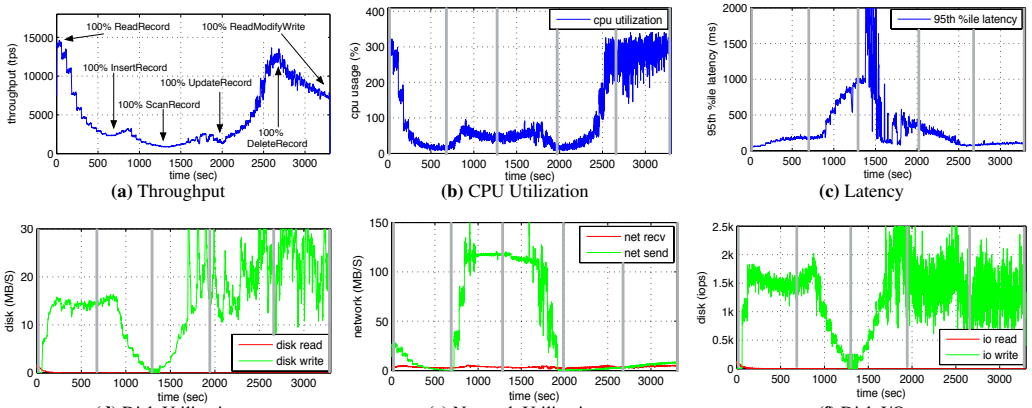
that price point does not yield in our tests significant throughput improvements. We suspect that this is because at high throughput rates the disk subsystem becomes the main bottleneck (and larger number of cores and more RAM are essentially not utilized), though we can only speculate on that point since this is an OS statistic that OLTP-Bench is unable to retrieve from a DBaaS (we will show next that for DBMSs we deploy this statistic is available).

Fig. 2 shows different results for executing the Wikipedia benchmark on Amazon RDS. Irrespective of the differences in absolute values with YCSB, which are dependent on the actual workload, the results indicate that the Wikipedia benchmark obtains better throughput and latency for the larger, more expensive instances. We suspect that since Wikipedia's workload is read-intensive, the CPU is the main bottleneck because the benchmark's working set fits in memory. As for the price/performance ratio, the results suggest that the XL-HM instance is the best choice for this workload. Although the 2XL-HM and 4XL-HM instances provide better performance, the additional cost incurred by the more expensive machines is unlikely to outweigh their performance advantage for most customers.

## 4.2 Evolving Workload Mixtures

We now test OLTP-Bench's ability to smoothly evolve the transaction mixture during an experiment. We choose YCSB as our target workload since it is composed of a series of simple transactions that each performs a specific type of operation (as opposed to the more complex transactions in other benchmarks that execute a diverse set of queries). The default scale factor for YCSB for this and the previous experiment is 1.2M tuples. Using MySQL, we first run the system at its maximal throughput using a single transaction type from YCSB (`ReadRecord`). Then, over a 10 minute period, we gradually transition to a workload mixture consisting of 100% of the next transaction type (`InsertRecord`), by changing the ratio by 10% every minute. We repeat this process for the four remaining transaction types in YCSB.

The graphs in Fig. 3 show the throughput and 95th percentile latency, and several resource metrics to indicate how different transactions types stress different resources. Each figure is annotated at the time when the next transition is started in the workload mix-




**(a)** Throughput **(b)** CPU Utilization **(c)** Latency

**(d)** Disk Utilization **(e)** Network Utilization **(f)** Disk I/O

**Figure 3:** Evolving Mixture – MySQL running YCSB demonstrating evolving mixture of transactions.

ture. These results are revealing of the underlying type of operation performed by the transactions. For example, Fig. 3b shows that the `ReadRecord` transactions are CPU-intensive due to parsing and in-memory index look-ups, while the `ScanRecord` transactions show heavy network I/O in Fig. 3e and longer latencies in Fig. 3c due to MySQL becoming network-bound. Fig. 3d shows that the transactions that write data, such as `InsertRecord` and `UpdateRecord`, cause the DBMS to become disk-bound. Deletes are also disk-intensive, but since less data is written for each operation (only undo logs), throughput and CPU load are higher.

The throughput of the `ReadModifyWrite` transaction in Fig. 3a is particularly interesting. This transaction performs the same update operation as the `UpdateRecord` transaction (but with a select query before the update), yet it achieves a significantly higher throughput. This is because the `DeleteRecord` phase removes a large number of tuples right before the `ReadModifyWrite` phase, and as a result a large percentage of the `ReadModify-Write` transactions are trying to modify non-existing tuples. A simple reordering of the phases in the experiment would correct this issue, but we left it as is to illustrate how the detailed OS resource and performance metrics helped us track down this problem.

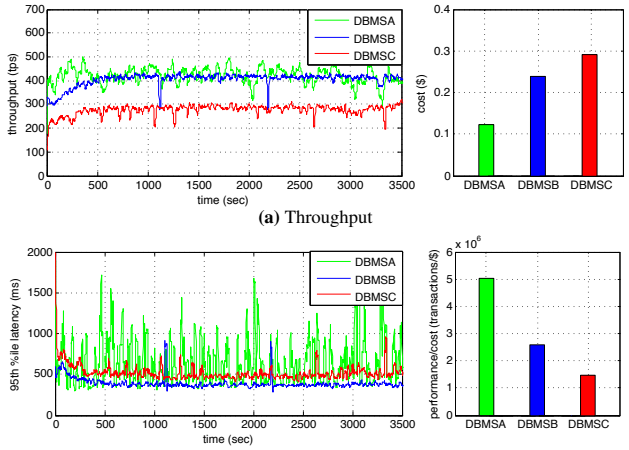
## 4.3 Comparing DBaaS Providers

In this final experiment, we compare all of the major DBaaS offerings using the SEATS benchmarks. Since network latency is an important factor for OLTP workloads, we ran all of the workers on virtual machines hosted in the same datacenter region but in a different data center from the DBMSs under test to ensure fairness between the DBaaS providers. We anonymize the providers name to prevent direct comparisons. The results in Fig. 4 show that DBMS-A has an erratic latency behavior but offers the best performance-vs.-cost ratio. DBMS-B achieves a steadier throughput and lowest latency for a higher price. Again, this is a simple yet convincing example of how proper benchmarking can help better understand the trade-offs between different aspects of cloud-based deployments.

Even the small sample of experiments we present is sufficient to highlight how DBaaS and the cloud, are a new terrain that can be understood by users only through careful benchmarking.

# 5. CONCLUSIONS

Benchmarking has alway been central for high-performance and efficient data management system design, deployment and maintenance. The advent of cloud-based data management solutions with their highly shared resources, non-user controlled system tuning, and novel pricing schemes further exacerbate the need for careful benchmarking.




**(a)** Throughput

**(b)** Latency

**Figure 4:** DB as a Service: Performance tradeoff across competing DBaaS offerings for SEATS across data-centers (opaque names and different workload to preserve anonymity of commercial vendors.)

The OLTP-Bench effort we presented in this extended-abstract is our answer to the lack of available workloads and of an appropriate testing infrastructure that made so far benchmarking a costly and error-prone activity. The larger campaign of testing from which we presented some samples convinced us that the current combination of workloads and the many features of the infrastructure can be of great help in many practical situations and help increase repeatability and ease of comparison across research results, and help practitioners to cope with the wild environment we call the cloud. What we provide today is an open-source extensible infrastructure and the initial critical mass of workloads that we hope will foster a community interest and development.

# 6. REFERENCES

[1] C. Curino, E. Jones, R. A. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan, and N. Zeldovich. Relational Cloud: A Database Service for the Cloud. In *CIDR*, pages 235–240, 2011.

[2] A. Pavlo, E. P. Jones, and S. Zdonik. On predictive modeling for optimizing transaction execution in parallel OLTP systems. *Proc. VLDB Endow.*, 5:85–96, October 2011.

[3] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz. Runtime measurements in the cloud: Observing, analyzing, and reducing variance. *PVLDB*, 3(1), 2010.