OLTP-Bench: An Extensible Testbed for Benchmarking Relational Databases

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

Andrew Pavlo
Carnegie Mellon University, USA
pavlo@cs.cmu.edu

Carlo Curino
Microsoft Corporation, USA
ccurino@microsoft.com

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

ABSTRACT

Benchmarking is an essential aspect of any database management system (DBMS) effort. Despite several recent advancements, such as pre-configured cloud database images and database-as-a-service (DBaaS) offerings, the deployment of a comprehensive testing platform with a diverse set of datasets and workloads is still far from being trivial. In many cases, researchers and developers are limited to a small number of workloads to evaluate the performance characteristics of their work. This is due to the lack of a universal benchmarking infrastructure, and to the difficulty of gaining access to real data and workloads. This results in lots of unnecessary engineering efforts and makes the performance evaluation results difficult to compare. To remedy these problems, we present OLTP-Bench, an extensible "batteries included" DBMS benchmarking testbed. The key contributions of OLTP-Bench are its ease of use and extensibility, support for tight control of transaction mixtures, request rates, and access distributions over time, as well as the ability to support all major DBMSs and DBaaS platforms. Moreover, it is bundled with fifteen workloads that all differ in complexity and system demands, including four synthetic workloads, eight workloads from popular benchmarks, and three workloads that are derived from real-world applications. We demonstrate through a comprehensive set of experiments conducted on popular DBMS and DBaaS offerings the different features provided by OLTP-Bench and the effectiveness of our testbed in characterizing the performance of database services.

1. INTRODUCTION

Performance analysis and tuning is one of the more difficult aspects of data management. It is especially challenging for complex systems that execute highly-concurrent workloads on large-scale datasets, because there are many factors that influence performance. We suspect, in fact, that the recent success of distributed key-value storage systems [13] is at least partially due to the difficulty of understanding and predicting the performance of relational DBMSs for these execution environments.

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/3.0/. Obtain permission prior to any use beyond those covered by the license. Contact copyright holder by emailing info@vldb.org. Articles from this volume were invited to present their results at the 40th International Conference on Very Large Data Bases, September 1st - 5th 2014, Hangzhou, China.
Proceedings of the VLDB Endowment, Vol. 7, No. 4
Copyright 2013 VLDB Endowment 2150-8097/13/12.

To overcome this problem, it is imperative that application developers use a testing environment that is *stable*, *controlled* and *repeatable* [19]. In the context of DBMSs, this is achieved through the use of a *benchmark* that allows one to measure key performance metrics of a system under stress conditions. A benchmark consists of a sample *data set* and a corresponding *workload* that is representative of the target application for the system. A good benchmark provides insight to users and enables them to (1) validate alternative options (e.g., by experimenting with different systems or configurations), (2) collect performance metrics and compare them with real-world requirements, and (3) explore the causes of performance bottlenecks. From these requirements emerges the need for a comprehensive *testbed* that supports both a large number of database systems and a wide range of benchmarks that capture the essence of an important set of applications. Although a number of benchmarks have been proposed in the past [10, 16], to the best of our knowledge such an extensive testbed is not available today.

The work that we present in this paper represents our effort to tackle this problem and to contribute to better repeatability and easier comparison of results for evaluating DBMS performance. We present *OLTP-Bench,* an extensible, open-source testbed for benchmarking DBMSs using a diverse set of workloads [2]. This effort is motivated by our own laborious experiences in setting up experimental evaluation frameworks for previous research projects. The testbed is capable of (1) driving relational DBMSs via standard interfaces, (2) tightly controlling the transaction mixture, request rate, and access distribution of the workload, and (3) automatically gathering a rich set of performance and resource statistics.

In addition to the testbed itself, we implemented 15 benchmarks in OLTP-Bench. We believe that the combination of a well-designed testbed with a rich family of benchmarks is a valuable asset. In summary, our contributions include:

- An automated and extensible framework to setup, run, and analyze the results of DBMS performance experiments with controlled and repeatable settings.
- Both real datasets and synthetic generators, along with their accompanying workloads, that span many interesting OLTP/Web applications, all implemented in the same framework.
- Experimental findings about popular DBMSs and DBaaS offerings derived from hundreds of experiments.

The rest of this paper is organized as follows: we start by discussing the motivation for this project in Section 2, followed by an introduction to the OLTP-Bench architecture and its main features in Section 3. Section 4 is dedicated to the description of the data and workloads that are currently bundled with OLTP-Bench. We then demonstrate how OLTP-Bench can be leveraged in many different benchmarking scenarios in Sections 6 and 7. Finally, we present our conclusions and discuss how we plan to extend our work and foster a community-driven effort in Sections 9 and 10.

2. MOTIVATION

In the early 1990s, Jim Gray edited a compendium that discussed popular benchmarks and various metrics (e.g., price vs performance) for evaluating transaction processing systems [19]. Gray argued both for generic benchmarks, which are useful in comparing the general performance of several systems, and domain-specific benchmarks, since "performance varies enormously from one application domain to another." For the latter, a meaningful benchmark is one that is *relevant*, *portable*, *scalable*, and *simple*.

Given these guidelines, one fundamental design principle of the OLTP-Bench project is that it does not impose any fixed set of configuration rules or pre-defined metrics of success for the benchmarks. We do not want to restrict the applicability of our framework and workloads to any specific context, since we believe that it is impossible to devise any single set of rules that will be applicable and relevant for all future deployment and execution scenarios. For example, novel hardware configurations are likely to require different database sizes and access distributions, while new execution models may require a different number of concurrent connections, latency requirements, and transaction mixtures. Hence, we contend that such standardizations are often biased towards the opinions of the proposers, and are likely to lead to design decisions that are only aimed at "scoring well" according to synthetic metrics of success. Nevertheless, having an immediate appreciation of how a system achieved is important and can be extracted from the classical throughput and latency metrics.

Others have called for the diversification of the data sets used for database research by encouraging more public and challenging workloads [39]. Existing benchmarking frameworks, however, only support a small number of workloads [16, 24] or a single DBMS [3, 5]. Our framework improves on previous efforts by supporting a greater selection of both benchmarks and DBMSs. We also follow the lead of [16] in leaving competition rules unspecified and let the users decide what the most suitable use cases are given their execution environment.

As mentioned above, this effort is motivated by the challenges that we faced when setting up performance evaluation campaigns for our previous projects [17, 25], as well as from extensive discussions with several other database developers and experienced researchers. From these interactions, we gathered the following list of requirements for a modern database testbed:

- **(R1)** **Transactional Scalability**: the ability to scale to high throughputs, without being restricted by clients;
- **(R2)** **Flexible Workload Generation:** the ability to generate workloads for open, closed, and semi-open loop systems;
- **(R3)** **Fine-Grained Rate Control:** the ability to control request rates with great precision (since even small oscillations of the throughput can make the interpretation of results difficult);
- **(R4)** **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 to simulate real-life events and evolving workloads;
- **(R5)** **Synthetic and Real Data & Workloads:** the need to handle both synthetic and real data sets and workloads, to avoid compromises between real-life tests and ease of scalability;

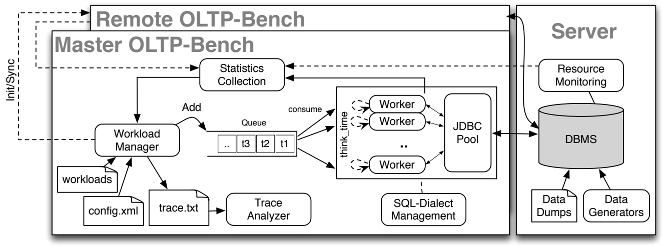


Figure 1: The architecture of the OLTP-Bench framework; On the left-hand side, the client-side driver handles workers and generates throttled workloads according to configuration provided by the user. On the right-hand side, the DBMS server monitor gathers resource utilization statistics.

- **(R6)** **Ease of Deployment:** the ability to deploy experimental settings and run experiments easily on a variety of DBMSs and DBaaSs using an integrated framework;
- **(R7)** **Extensibility:** the ability to extend the testbed to support different benchmark configurations or to integrate a new benchmark with a small engineering effort;
- **(R8)** **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;
- **(R9)** **Assisted Statistics Visualization:** the ability to render graphs for collected statistics semi-automatically, to minimize the time-to-insight during experiments.
- **(R10)** **Repeatability and Verification:** the ability to rerun experiments under specific circumstances to verify results, check for completeness and consistency.

3. THE OLTP-Bench TESTBED

This section provides an overview of the architecture of the OLTP-Bench testbed and discusses how it is designed to meet the identified requirements in Section 2. OLTP-Bench works with any single-node DBMS, distributed DBMS (e.g., NewSQL), and DBaaS system that supports SQL through JDBC. Fig. 1 shows the architecture of our framework; its two main components are (1) the client-side benchmark driver and (2) a server-side module. OLTP-Bench is written entirely in Java, including all of the built-in benchmarks. The framework itself has almost no impact on the DBMS's performance. The client-side portion is small and portable (less than 5MB, excluding workload traces and sample data sets). The framework has been tested and deployed on a variety of Unix-like platforms.

3.1 Workload Manager

The benchmark driver is responsible for controlling the client workers that execute queries on the target system and gather performance statistics from them. The centralized *Workload Manager* reads the user-provided benchmark configuration file ("config.xml") that specifies the parameters used in the benchmark invocation, including the size of the connection pool (i.e., corresponding to the number of parallel connections), the exact composition of the workload, the desired transaction throughput, and the duration. All of these parameters are specified for each "phase" of the experiment, enabling evolving behaviors.

At runtime, the Workload Manager instantiates one or more *Worker* threads, and populates a request queue that is consumed by them. As described in Section 3.4, these threads may run on the same node as the Workload Manager or may run on other machines. The Workers' request queue offers a trade-off between requirements **R1**, **R2**, **R3**, and **R4**; 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 lightweight enough to achieve good scalability. In a simple test using the YCSB benchmark with a main-memory DBMS, we were able to generate 12.5k transactions per second per Worker thread.

The Workers connect to the target DBMS using JDBC and iteratively pull tasks from the request queue. For each new transaction, the Worker invokes the corresponding transaction's control code (i.e., program logic with parameterized queries) and either commits or aborts the transaction. The Workload Manager creates requests for the Workers using either a synthetic load generator or from "replaying" a pre-existing execution trace (**R5**). For example, for the Wikipedia benchmark, we use a trace from the actual website.

3.2 Workload Generation

OLTP-Bench supports three different system models for Workers to invoke transactions: (1) closed-loop, (2) open-loop, and (3) semi-open-loop [32]. Allowing users to choose which model to use in their testing enables them to explore different application scenarios and simulate several real-world scenarios (**R2**). In closed-loop testing, OLTP-Bench initializes a fixed number of Workers that repeatedly issue transactions with a random think time between each request. With the open-loop execution setting, the rate at which requests are invoked follows a stochastic process. Lastly, under a semi-open policy, the system acts essentially as an open-system with the difference that the Worker pauses for a random think time before submitting a new transaction. This simulates real-world workloads where clients do not immediately submit a new request as soon as their previous query is answered.

In addition to the aforementioned features, OLTP-Bench can handle dynamic changes to the number of Workers and transaction weights over the phases to allow more complex modeling. OLTP-Bench can also drive multiple benchmarks at the same time to test the ability of a DBMS to balance the workload among different databases or multiple tenants.

3.3 SQL Dialect Translation

In order to support multiple DBMSs in a single framework, we developed a special sub-component, called the *SQL-Dialect Manager*, that rewrites the pre-defined queries for each transaction from SQL-92 to the native format of the target DBMS (**R6**)[1]. Along with the SQL-Dialect Manager, OLTP-Bench uses a universal internal catalog for each benchmark (regardless of the target DBMS) to extract additional information about the benchmark's tables and columns. This allows the benchmark's utility code and transaction control code to be ported to new DBMSs with minimal effort (**R7**). The current version of OLTP-Bench supports a number of commercial systems, including MSSQL, Oracle, and DB2, as well as popular open-source systems, such as MySQL and PostgreSQL.

3.4 Distributed Clients

In order to saturate high-performance distributed DBMSs with enough transaction requests, OLTP-Bench supports deploying workers across multiple machines [17, 25]. Multiple remote OLTP-Bench instances are spawned by the central Workload Manager via SSH (**R1**). The Manager then sends commands for starting and stopping the benchmark trial over a network socket. At the end of the benchmark run, it retrieves the results from the client nodes and merges them into a single output file.

[1]More complex and non-standard SQL must be manually loaded into OLTP-Bench. We defer the problem of automatically translating SQL to the appropriate dialect as future work.

Class	Benchmark	Application Domain
Transactional	**AuctionMark**	On-line Auctions
	CH-benCHmark	Mixture of OLTP and OLAP
	SEATS	On-line Airline Ticketing
	SmallBank	Banking System
	TATP	Caller Location App
	TPC-C	Order Processing
	Voter	Talent Show Voting
Web-Oriented	**Epinions**	Social Networking
	LinkBench	Social Networking
	Twitter	Social Networking
	Wikipedia	On-line Encyclopedia
Feature Testing	**ResourceStresser**	Isolated Resource Stresser
	YCSB	Scalable Key-value Store
	JPAB	Object-Relational Mapping
	SIBench	Transactional Isolation

Table 1: The set of benchmarks supported in OLTP-Bench.

3.5 Runtime Results & Statistics Collection

As the workers execute transactions and receive results from the DBMS, they also collect detailed statistics about the workload. The workers maintain a compact in-memory representation of these statistics, thereby avoiding the potential risk of blocking due to disk writes outside of the DBMS (**R1**, **R8**). These statistics are continuously forwarded to the Workload Manager, which aggregates them according to the experimental setup, and then writes both the raw statistics and their aggregated version into the benchmark output file. A separate component, called the *Trace Analyzer*, reads these results and generates graphs for the user (**R9**).

We monitor the resource utilization on the server side using our own extended version of the DSTAT libraries[2]. This light-weight server monitoring component captures statistics from the OS (i.e., CPU, RAM, and I/O activity) and from the DBMS itself whenever possible (i.e., resource consumption) with minimal impact on performance. At the end of the experiment, these statistics are sent to the Trace Analyzer and are automatically aligned with the data collected by the clients based on timestamps. The resource monitoring is specific to the operating system and the DBMS (since no generic API is available for this task).

Finally, given an (optional) file containing authoritative results, the system compares the results obtained through the experiments to the expected results (**R10**).

4. BENCHMARK DATA & WORKLOADS

We now provide an overview of the benchmarks currently implemented in OLTP-Bench. Table 1 gives the application domain for each benchmark and Table 2 shows their profile statistics. The size of each benchmark's database is configurable by the administrator and the working set size is automatically self-scaling.

Jim Gray's effort in the early 1990s (see Section 2) evaluated six different benchmarks for transaction processing systems. Since then, the range of applications requiring transaction support has grown with the emergence of Web-based application domains like collaborative editing, social networking, or on-line reservations. Given the multiplicity and diversity in OLTP and Web-based applications, we do not claim that this initial set of benchmarks is exhaustive nor that their classes are definitive. We believe, however, that each of the following benchmarks is useful in modeling a specific application domain. We now describe the workloads in detail and categorize them into three groups: transactional workloads, Web-oriented workloads (including social network applications), and workloads designed to facilitate isolated feature testing.

[2]http://dag.wieers.com/home-made/dstat/

	Tables	Columns	Indexes	Txns	Read-Only Txns
AuctionMark	16	125	14	9	55.0%
CH-benCHmark	12	106	3	27	54.0%
Epinions	5	21	10	9	50.0%
JPAB	7	68	5	4	25.0%
LinkBench	3	17	1	10	69.05%
ResourceStresser	4	23	0	6	33.3%
SEATS	10	189	5	6	45.0%
SIBench	1	2	1	2	50.0%
SmallBank	3	6	4	6	15.0%
TATP	4	51	5	7	40.0%
TPC-C	9	92	3	5	8.0%
Twitter	5	18	4	5	0.9%
Voter	3	9	3	1	0.0%
Wikipedia	12	122	40	5	92.2%
YCSB	1	11	0	6	50.0%

Table 2: Profile information for the benchmark workloads.

4.1 Transactional Benchmarks

This category contains traditional OLTP benchmarks characterized by write-heavy transactions with complex relations.

4.1.1 AuctionMark

AuctionMark is an OLTP benchmark that models the workload characteristics of an on-line auction site [6]. It consists of 10 transactions, one of which is executed at a regular interval to process recently ended auctions. The database and workload properties are derived from information extracted from a well-known auction site, as well as from another benchmark [28]. The user-to-item ratio follows a Zipfian distribution, which means that there are a small number of users that sell a large portion of the items. The total number of transactions that target each item is temporally skewed, as items receive more activity as the auction approaches its closing.

This benchmark is useful for measuring the performance of a DBMS for workloads with a large number of tables that cannot easily be denormalized. Another notable feature is that the amount of work performed by each transaction is non-deterministic. For example, a new bid on an item may require updating multiple records depending on whether it is greater than the current highest bid.

4.1.2 CH-benCHmark

This is a mixed workload derived from TPC-Cand TPC-H [15]. It is useful to evaluate DBMSs designed to serve both OLTP and OLAP workloads. The implementation leverages the ability of OLTP-Bench to run multiple workloads. It uses our built-in implementation of TPC-C along with 22 additional analytical queries.

4.1.3 SEATS

The SEATS benchmark models an airline ticketing system where customers search for flights and make on-line reservations [35]. It consists of eight tables and six transaction types. Approximately 60% of the transactions are read-only (e.g., customers searching for open seats), while the other 40% involve creating, updating, and deleting reservation records.

The benchmark is designed to emulate a back-end system that processes requests from multiple applications that each provides disparate inputs. Thus, many of its transactions use secondary indexes or foreign-key joins to find the primary key of a customer's reservation record. For example, customers may access the system using various credentials, including their frequent flyer number, their customer account number, or their login name.

4.1.4 SmallBank

This workload models a banking application where transactions perform simple read and update operations on customers' accounts [11]. All of the transactions involve a small number of tuples. The transactions' access patterns are skewed such that a small number of accounts receive most of the requests. We also extended the original SmallBank implementation to include an additional transaction that transfers money between accounts.

4.1.5 TATP

The TATP benchmark is an OLTP application that simulates a caller location system used by telecommunication providers [40]. The benchmarks consists of four tables, three of which are foreign key descendants of a single "root" table. All seven transactions in TATP reference tuples using either the root's primary key or a separate unique identification string. The transactions that are only provided with the non-primary key identifier use a secondary index to find the root record that corresponds to this identifier.

The transactions in this benchmark are more lightweight than the ones in the other benchmarks supported in OLTP-Bench. All of TATP's transactions contain only 1-3 queries and 80% of them are read-only. TATP does include, however, a larger number of transactions that abort due to assertions in their control code. As such, this benchmark provides a useful workload scenario for measuring a DBMS's ability to run non-conflicting transactions concurrently.

4.1.6 TPC-C

The TPC-C benchmark is the current industry standard for evaluating the performance of OLTP systems [36]. It consists of nine tables and five procedures that simulate a warehouse-centric order processing application. All of the transactions in TPC-C provide a warehouse id as an input parameter that is the ancestral foreign key for all but one of TPC-C's tables. The number of `NewOrder` transactions executed per second is often used as the canonical measurement for the throughput of a DBMS. TPC-C's transactions are more complex and write-heavy than in other benchmarks (e.g., under the default settings, 92% of TPC-C's issued transactions modify tables). One interesting aspect of TPC-C is that if the number of warehouses in the database is sufficiently small, then the DBMS will likely become lock-bound.

OLTP-Bench's version of TPC-C is a "good faith" implementation, although we omit the "thinking time" for workers. This means that each worker issues transactions without pausing, and thus only a small number of parallel connections are needed to saturate the DBMS. This mitigates the need to increase the size of the database with the number of concurrent transactions[3].

4.1.7 Voter

The Voter workload is derived from the software system used to record votes for a Japanese and Canadian television talent show. As users call in to vote on their favorite contestant during the show, the application invokes transactions that update the total number of votes for each contestant. The DBMS records the number of votes made by each user up to a fixed limit. A separate transaction is periodically invoked to compute vote totals during the show.

This benchmark is designed to saturate the DBMS with many short-lived transactions that all update a small number of records.

4.2 Web-Oriented Benchmarks

The following set of benchmarks model Web-based applications. These workloads feature social networks with graph traversal operations on many-to-many relationships with non-uniform access. These benchmarks are designed to mimic real-world applications using publicly available traces and data dumps.

[3]In the official version of TPC-C, each worker acts on behalf of a single customer account, which is associated with a warehouse.

4.2.1 Epinions

This benchmark is derived from the Epinions.com consumer review website. It uses data collected from a previous study [23] together with additional statistics extracted from the website. This workload is centered on users interacting with other users and writing reviews for various items in the database (e.g., products). It consists of nine different transactions, of which four interact with user records only, four interact with item records only, and one interacts with all of the tables in the database. Users have both an n-to-m relationship with items (i.e., representing user reviews and ratings of items) and an n-to-m relationship with users.

This workload emerged from one of the original "social networking" websites and thus provides an interesting challenge for relational DBMSs. It is similar to the Twitter benchmark, except that its many-to-many relationships are traversed using SQL joins (as opposed to application-side joins), and it has more complex interactions between its tables.

4.2.2 LinkBench

This synthetic benchmark was developed by Facebook to evaluate systems running a workload similar to their MySQL production deployment [8]. The social graph is synthetically generated at configurable scale while keeping the properties of the real social graph. The workload is based on real traces of queries executed on the production system.

4.2.3 Twitter

The Twitter workload is inspired by the popular micro-blogging website. In order to provide a realistic benchmark, we obtained an anonymized snapshot of the Twitter social graph from August 2009 that contains 51 million users and almost 2 billion "follows" relationships [14]. We created a synthetic workload generator that is based on an approximation of the queries/transactions needed to support the application functionalities as we observe them by using the web site, along with information derived from a data set of 200,000 tweets. Although we do not claim that this is a precise representation of Twitter's system, it still reflects its important characteristics, such as heavily skewed many-to-many relationships.

4.2.4 Wikipedia

This workload is based on the popular on-line encyclopedia. Since the website's underlying software, MediaWiki, is open-source, we are able to use the real schema, transactions, and queries as used in the live website. This benchmark's workload is derived from (1) data dumps, (2) statistical information on the read/write ratios, and (3) front-end access patterns [38] and several personal email communications with the Wikipedia administrators. Although the total size of the Wikipedia database exceeds 4TB, a significant portion of it is historical or archival data (e.g., every article revision is stored in the database). Thus, the working set size at any time is much smaller than the overall data.

We extracted and modeled the most common operations in Wikipedia for article and "watchlist" management. These two operation categories account for over 99% of the actual workload executed on Wikipedia's underlying DBMS cluster. The combination of a large database (including large secondary indexes), a complex schema, and the use of transactions makes this benchmark invaluable to test novel indexing, caching, and partitioning strategies.

4.3 Feature Testing Benchmarks

OLTP-Bench's third category of benchmarks are intended to test individual features of a system. These workloads, also known as micro-benchmarks, are simpler and more lightweight than the benchmarks found in the other two categories.

4.3.1 JPAB (Object Relation Mapping)

Since object-relational mapping tools are often used in DBMS-based applications, especially in enterprise settings, we ported the Java Persistence API Performance Benchmark (JPAB) to our framework [1]. This workload represents a large class of enterprise applications that have several properties that are unique to ORMs. For example, many ORM implementations generate unoptimized bursts of small reads in order to chase object pointers. As such, this workload can be used to test various improvements in both the application and DBMS-level for this type of application.

4.3.2 ResourceStresser

In contrast to most of the other benchmarks in OLTP-Bench that emulate existing systems or common application patterns, we developed ResourceStresser as a purely synthetic benchmark that can create isolated contention on system resources. Each of the benchmark's transaction imposes some load on three specific resources: CPU, disk I/O, and locks. As an example, the CPU-intensive transactions repeatedly execute SQL encryption functions on small amounts of data. Using these transactions as building blocks, one can control the workload mixture in the framework to simulate diverse kinds of applications and execution scenarios.

4.3.3 SIBench

SIBench is a microbenchamark designed to explore snapshot isolation in DBMSs [12]. It contains a single key/value table and two transactions that fetch the minimum value of a column or increment a single value of an entry. This workload creates a situation where the DBMS must resolve read-write conflicts while also stressing the CPU by scanning the table for the minimum value.

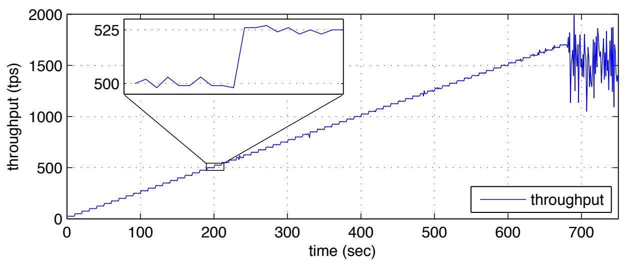
4.3.4 YCSB

The Yahoo! Cloud Serving Benchmark (YCSB) is a collection of micro-benchmarks that represent data management applications whose workload is simple but requires high scalability [16]. Such applications are often large-scale services created by Web-based companies. Although these services are often deployed using distributed key/value storage systems, this benchmark can also provide insight into the capabilities of traditional DBMSs.

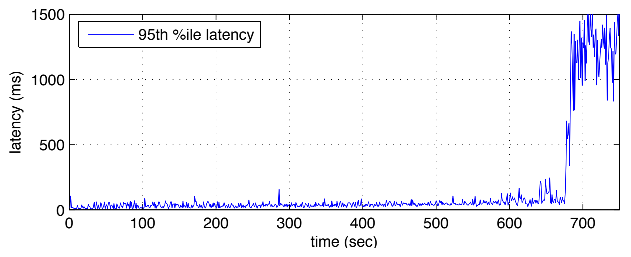
The YCSB workload contains various combinations of read/write operations and access distributions that match products inside Yahoo! It is representative of simple key-value store applications. The benchmark has been leveraged in previous studies for exploiting the trade-offs between availability/consistency/partition tolerance, and more generally to showcase storage engines and caching results (e.g., improving the throughput of random writes).

5. EXPERIMENTAL DEPLOYMENT

We now turn to two series of experiments that we performed to demonstrate the capabilities of OLTP-Bench. In Section 6, we first showcase the key features of the OLTP-Bench testbed and demonstrate the relevance of the workloads we have implemented. We then present experiments on DBaaS systems in Section 7 that examine the variance in service quality, and the metrics that OLTP-Bench computes. We also use OLTP-Bench to evaluate the DBaaSs with respect to price and instance selection for a given scenario. We use a total of three different DBaaS offerings from two different vendors: two from Amazon's RDS and the other from Microsoft's SQL Azure platform. Due to the commercial license agreements for both vendors, we anonymize the service names in this section



(a) Throughput



(b) Latency

Figure 2: Rate Control – MySQL running Wikipedia demonstrating precise control of request rates.

using different identifiers than in Section 6 to avoid indirect inferences. Since these DBaaSs are provided as a "managed" solution, there are no configuration or tuning parameters to consider. Each DBaaS is used with the default configuration without adding any special features available for an additional fee.

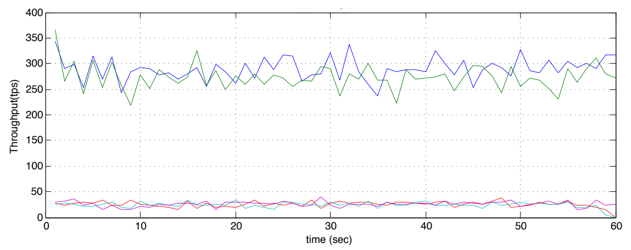
As reported in [30], the variability of cluster conditions in "noisy" cloud computing environments can significantly affect benchmark results. To mitigate such problems, we ran all of our experiments with restarting instances as little as possible, and executed the benchmarks multiple times and averaged the results. All RDS/EC2 instances were located in the *US-East* region, while for Azure we used the *US-West* region.

In each experiment, both the throughput and latency measurements are aggregated over one-second-windows. Latency is measured as the processing time of a transaction, from start to end, excluding internal queuing times. The client-side module logs the running times of all the transactions such that latency percentiles and averages can be computed a posteriori during the trace analysis phase. These experiments are not meant to show absolute differences between the DBMSs, but rather to show that OLTP-Bench supports features that are important in benchmarking OLTP and Web-based applications on DBMSs. Using these features together with the benchmarks enables one to understand the performance and scalability limits of different systems running in different application domains and on different infrastructures.

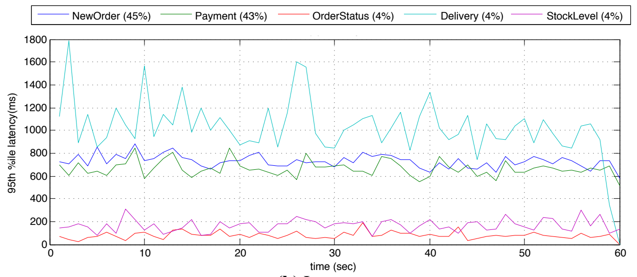
6. OLTP-Bench FEATURE EVALUATION

In this section we showcase the key features of theL OLTP-Bench testbed. Although OLTP-Bench supports different DBMS vendors, for the first experiments in this section we use a single DBMS type (MySQL) deployed on the Amazon EC2 platform. This allows us to compare the differences of several experiments for multiple workloads and highlight how OLTP-Bench's metrics helps in diagnosing some performance issues in typical scenarios. Unless specified, we used the XLarge instance type of EC2, and a connection pool of 200. The full set of configurations for these comparisons is available on the project's website [2].

6.1 Rate Control



(a) Throughput



(b) Latency

Figure 3: Multi-Class Reporting– MySQL running TPC-C showing the behavior of each of the five TPC-C's transaction types.

We begin with demonstrating OLTP-Bench's ability to control the request rates of transactions with fine-grained precision. We use MySQL as a reference DBMS and run the Wikipedia benchmark on a 100k article database starting at 25 transactions per second (tps) and growing by 25 tps every 10 seconds. We chose Wikipedia for this experiment because both its data and workload are heavily skewed (in size and access frequency, respectively), which makes them difficult to control at the granularity that we need. The data and workload trace used in all of the experiments for Wikipedia are generated synthetically based on real traces. We separately compared results achieved by MySQL under identical conditions for both synthetic and real data and traces, and observed 1.7% relative error for throughput and 1% for average latency—higher percentiles of latency and resource consumption metrics diverge more dramatically. This accuracy is sufficient for most scenarios. The real data and traces are available when higher accuracy is required.

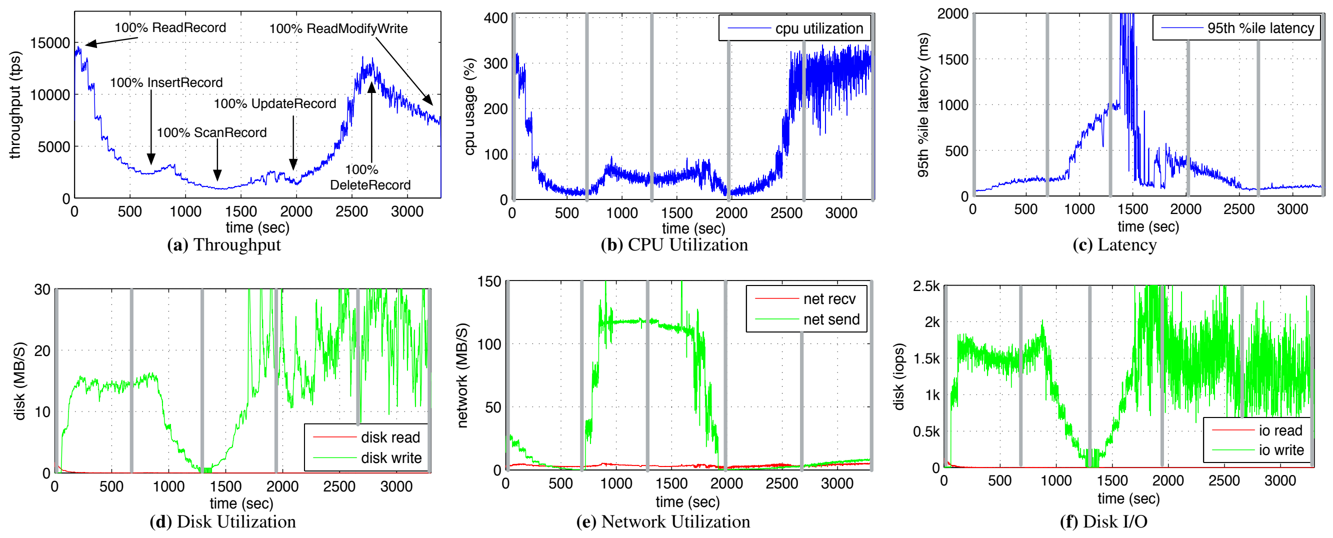
Fig. 2a shows that MySQL's throughput almost matches the target request rates until ∼680 seconds into the experiment, at which point the DBMS becomes overloaded. The zoomed-in portion of the graph illustrates the framework's ability to change the throughput in small increments, and to hold the throughput at a constant rate. The latency measurements in Fig. 2b also show that the 95th percentile latency is small until the framework hits saturation, at which point the latency increases to over one second.

6.2 Multi-Class Reporting

Being able to understand how a DBMS behaves with multi-class workloads is another important feature that our testbed offers. OLTP-Bench collects information on each transaction separately at runtime, and then automatically groups the results according to the transaction type for easy visualization. In this experiment, we ran the TPC-C benchmark at saturation and collect the throughput and latency measurements. The graphs in Fig. 3 show the performance breakdown per transaction type. Although the `NewOrder` and `Payment` transactions represent the majority of the transaction in the TPC-C workload, the `Delivery` transaction has the most significant impact on the overall system response time.

6.3 Evolving Workload Mixtures

We now test OLTP-Bench's ability to smoothly evolve the transaction mixture during an experiment. We choose YCSB as our tar-



(a) Throughput (b) CPU Utilization (c) Latency (d) Disk Utilization (e) Network Utilization (f) Disk I/O

Figure 4: Evolving Mixture – MySQL running YCSB demonstrating evolving mixture of transactions.

get workload since it is composed of a series of simple transactions that each perform 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 experiment and for following experiments 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. 4 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 mixture. These results are revealing of the underlying type of operation performed by the transactions. For example, Fig. 4b 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. 4e and longer latencies in Fig. 4c due to MySQL becoming network-bound. Fig. 4d 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 needs to be written per delete (only undo logs), the throughput and CPU load are higher.

The throughput of the `ReadModifyWrite` transaction in Fig. 4a is particularly interesting. This transaction performs the same update operation as the `UpdateRecord` transaction (but with a read query before the update), yet it achieves a significantly higher throughput. This is caused by the `DeleteRecord` phase, which removes a large number of tuples right before the `ReadModifyWrite` phase, and which results in a large percentage of the `ReadModifyWrite` transactions 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 metrics provided by OLTP-Bench helped us track down this problem.

6.4 Evolving Access Distributions

In this experiment, we test OLTP-Bench's ability to evolve the distribution of a workload's access patterns during execution. Simulating the formation and movement of hot spots in this manner enables users to carefully emulate some of the most elusive characteristics of real-life workloads, and thus investigate the system response under such conditions. In order to make the implications of this feature clear, we use the Twitter benchmark but configure its workload mixture to only execute transactions that retrieve a single tweet message via a primary key look-up. To drive this workload on MySQL, we used OLTP-Bench to generate a trace file that alternates between a uniform and a Zipfian distribution on the tweet's primary keys that are accessed by these transactions. We set the size of the workload's working set data to 4GB (roughly 20M tweets) and MySQL's buffer pool size to 2GB. This increases the likelihood that the DBMS will have to retrieve records from disk for the uniformly distributed transaction requests.

The results in Fig. 5 show that the MySQL satisfies most of the requests from cached requests in its buffer pool for a Zipfian-skewed access distribution, thereby achieving low-latency, high-throughput, and incurring minimal disk reads with higher CPU loads. The non-skewed access distribution, however, causes the DBMS to fetch data from disk, which in turn increases disk I/O.

6.5 Exploring Rich Metrics

Another important feature of OLTP-Bench is its ability to collect statistics from both the OS and DBMS directly during a benchmark invocation. We demonstrate how the rich metrics produced by OLTP-Bench can help administrators understand peculiar performance issues. As an example, we use a real problem that we encountered when running experiments for this paper using the YCSB benchmark on MySQL.

The graph in Fig. 6a shows the throughput of MySQL running YCSB for an hour. After ∼2600 seconds, the throughput suddenly begins to degrade for a six minute period. It then returns to its previous rate of ∼4000 tps. This behavior was repeatable and would always occur at the same point every time we ran this benchmark.

To investigate this phenomenon, we used two of the over 300 metrics collected by OLTP-Bench while a benchmark executes. The first metric, shown in Fig. 6b, is the amount of free space in the MySQL's buffer pool (measured as the number of pages, where each page is 16KB). The second metric, shown in Fig. 6c, is the number of physical reads performed by MySQL, corresponding to pages that were not in the buffer pool and therefore need to be fetched from disk. After an initial warm-up phase, the number

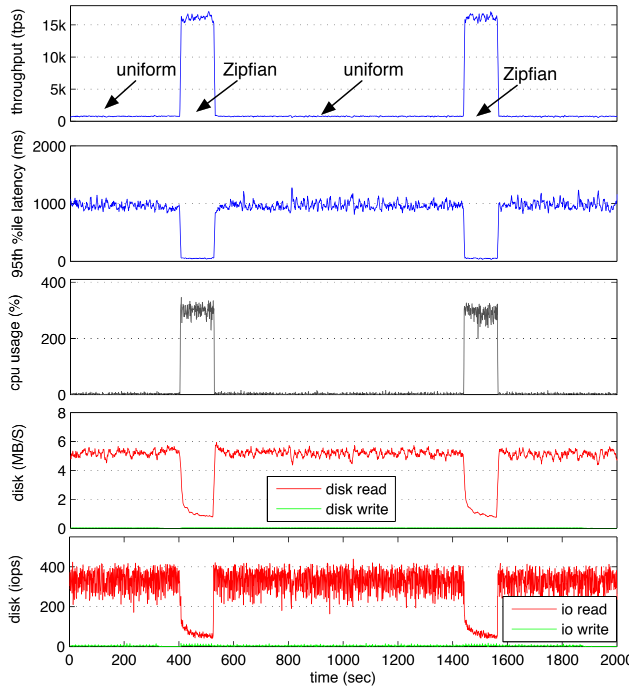


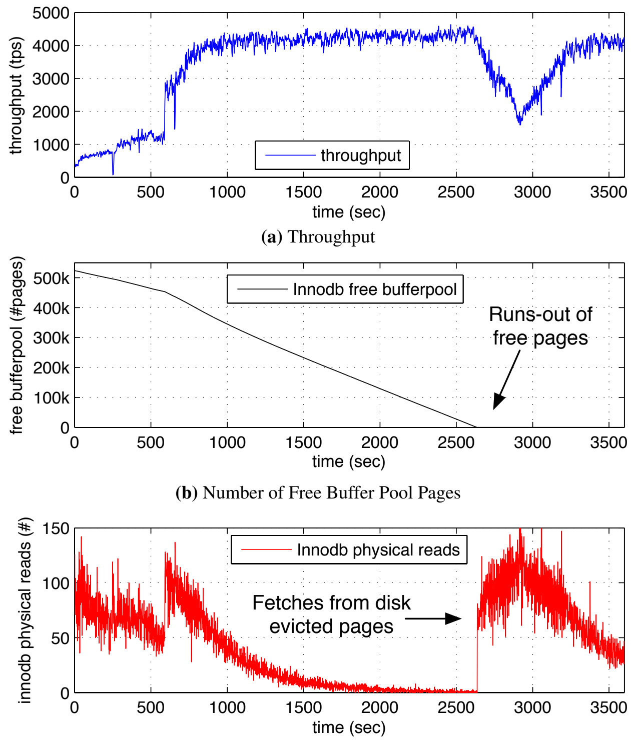
Figure 5: Evolving Access Distributions – MySQL running Twitter alternating random reads with uniform and Zipfian distributions.

of page reads is close to zero; this indicates that the benchmark's working set fits entirely in memory. After 2600 seconds, however, the graph in Fig. 6b shows that MySQL runs out of empty pages in the buffer pool and therefore must evict pages to accommodate new data. Since many pages get evicted all at once, it is likely that MySQL chooses to free up non-dirty pages (which can be dropped without a disk write) rather than pushing dirty pages to disk. As a result, the pages needed for new transactions are now only available from disk, which explains the increase in page reads shown in Fig. 6c. Eventually, the algorithms governing the eviction and those controlling dirty-pages write-backs converge to a more stable state, in which MySQL maintains the load effectively. The availability of a large variety of metrics and data, such as those automatically captured by OLTP-Bench, is invaluable when looking into specific issues like the one described above.

6.6 Multitenancy

We next use OLTP-Bench's ability to support multiple benchmarks simultaneously to explore how well MySQL is able to handle multiple workloads within the same DBMS instance. Such deployments are common in shared-hosting and virtualized environments. We execute the TATP, TPC-C, and Wikipedia benchmarks simultaneously and measure how well the system is able to handle the concurrent workloads. These benchmarks were chosen because they each have different characteristics.

We first load the three benchmarks' data sets into the target DBMS. We then start all three benchmark clients simultaneously running at throttled speeds: TATP at 5000 tps, TPC-C at 10 tps, and Wikipedia at 500 tps. These numbers were chosen from previous experiments showing that the three DBMSs were able to handle such workloads without becoming overloaded. After five minutes, we increase the requested throughput for TPC-C to push the DBMS to become disk-bound (due to the large number of writes in TPC-C), and observe the impact on the other two benchmarks before going back to the initial setting. Although we did not expect that each workload would be fully isolated from the effects of the other workloads, we wanted to observe the degree of fairness in each system's scheduler and the extent to which each workload is throttled.



(a) Throughput

(b) Number of Free Buffer Pool Pages

(c) Number of Physical Reads

Figure 6: Exploring Rich Metrics – MySQL running YCSB leveraging DBMS metrics to investigate an unexpected performance drop.

As shown in Fig. 7, MySQL sustains this combined workload elegantly in the first part of the experiment, fully serving the request rates imposed by the clients (normalized for the sake of presentation). MySQL provides good performance overall, albeit in a more skewed way, since Wikipedia is only slightly affected but TATP's throughput is reduced by 20%.

6.7 Distributed Clients

We demonstrate how OLTP-Bench can run multiple clients in parallel to overcome any client side bottleneck that might hinder the full exploitation of the server. We used the SEATS benchmark deployed in MySQL. We configured OLTP-Bench to use a fixed pool of 200 Worker threads that were deployed on EC2 Small instances. For each trial, we increase the number of client instances and evenly assign the Workers among them. The results in Fig. 8 show that using a single client machine achieves the lowest throughput and highest latency (for this benchmark). By distributing the client-side computation workload on more machines, we improve the overall throughput until reaching the saturated throughput of the server at five client instances.

6.8 Repeatability

Lastly, we demonstrate how OLTP-Bench can be used to validate or refute previous experimental results. Repeatability is difficult if the testbed is not publicly available, configuration parameters were not reported, or the hardware is difficult to acquire (e.g., outdated, expensive). We ran the SIBench workload with a hardware setting similar to the one used in [26]. Fig. 9 shows a comparison between Snapshot Isolation (SI) [26] versus the Serializable SI introduced in PostgreSQL v9.1. As previously reported in [12], the number of

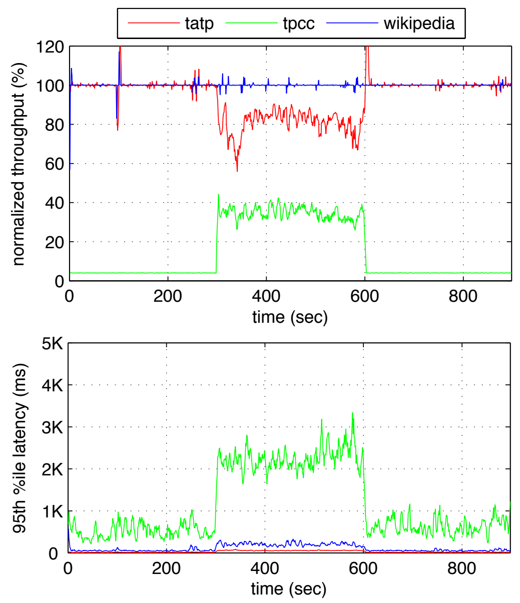


Figure 7: Multitenancy – MySQL running the TATP, Wikipedia, and TPC-C benchmarks at throttled speeds, followed by TPC-C running at its maximum speed.

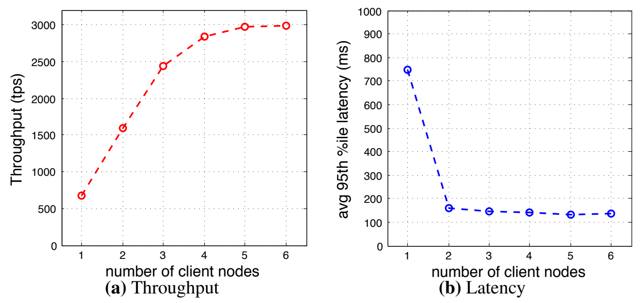


Figure 8: Distributed Clients – OLTP-Bench running a SEATS workload on 1–6 EC2-Small machines. The DBMS is MySQL running on EC2-XLarge instance with a connection pool of 200.

terminals increases the amount of lock contention in Serializable SI. But this information is omitted from [26], and thus we initially got inconsistent results. Only after contacting the authors did we learn that they used four terminals, which then made our experimental results in Fig. 9 match those originally reported in [26].

7. "DB-AS-A-SERVICE" EVALUATION

In contrast to self-managed DBMSs, users do not have access to the underlying system configuration used in a DBaaS platform. This makes it more difficult to detect and correctly diagnose performance problems, since the internals of the system are obfuscated.

To overcome this problem, we now show how our testbed helps administrators evaluate the performance of their databases deployed on DBaaS platforms. We begin in Section 7.1 by showing how OLTP-Bench can detect abnormal quality-of-service issues. We then compare the performance-vs-cost of different workloads executing on the same DBaaS platform in Section 7.3, as well as the performance-vs-cost of different DBaaSs executing the same workload in Section 7.3. Lastly, we show a side-by-side comparison of all the DBaaS platforms we tested in Section 7.4.

For all of the experiments in this section, OLTP-Bench was initiated from virtual machines hosted in the same data centers as their target DBaaS instances. We used 30 Worker threads and an fixed configuration for the workload mixture and target throughput rate.

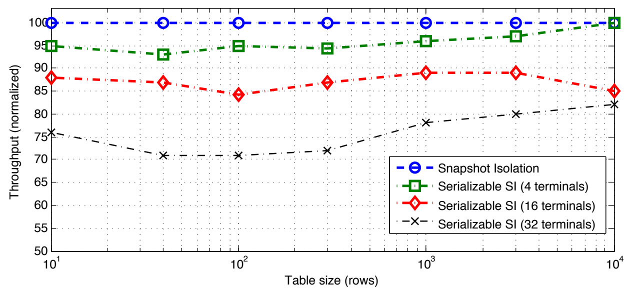


Figure 9: Repeatability – PostgreSQL running SIBench with varying number of rows and terminals. Here, Serializable SI transaction throughput is compared to SI as a percentage.

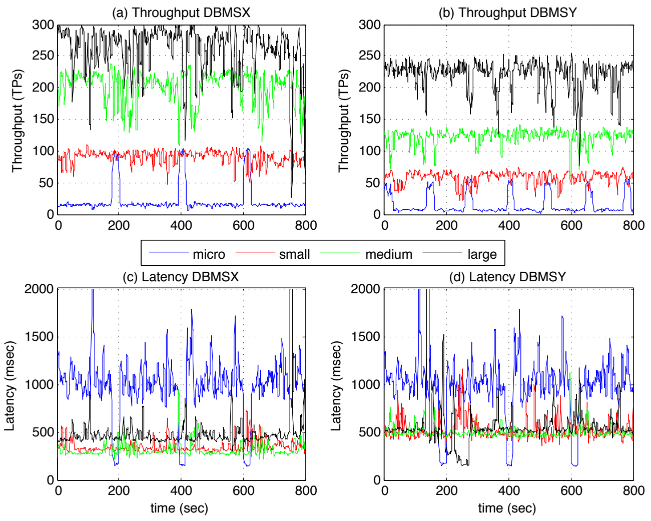


Figure 10: DBaaS – Performance on better instance types (micro, small, medium, large) on two DBaaS services.

7.1 Variance in the Cloud

We first present a set of experiments designed to showcase OLTP-Bench's ability to collect and visualize performance metrics that can be used to make vendors accountable for the quality of service on their platforms. Since a user has little or no control over the configuration parameters of a DBaaS, it is difficult to discern whether one is getting the expected service that they are paying for. In this case, our benchmarking methodology departs from traditional benchmarking tools in that we try to achieve statistically significant performance results in the cloud.

The results in Fig. 10 show the throughput and latency measurements of executing TPC-C on two DBaaS platforms. We ran the same workload at saturation on four different DBaaS instance types. There are two notable observations from these results. First, we see that the performance of micro instances show sporadic bursts. In addition, we observe that while better instances provide consistent throughput increases, the response time is noticeably better for the larger instances only.

Next, we explore the variability of services across the same instance types. Similar investigations were conducted on Amazon EC2 cloud in [29, 30] while in the present work we focus on DBaaSs offerings. For this experiment, we run TPC-C on four Medium instances. As shown in the results in Fig. 11, the performance of both DBMS-X and DBMS-Y are inconsistent over time and across different instances.

For both platforms, the results in Fig. 11 also show that some instances are more stable than others. To determine whether this is always the case, we ran TPC-C for over two hours at saturation

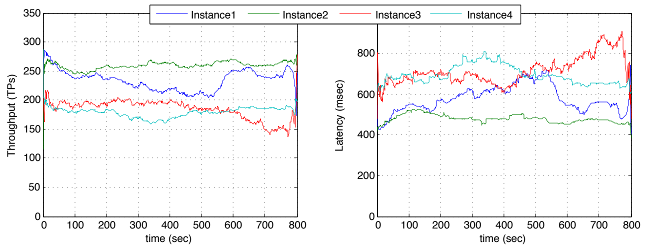
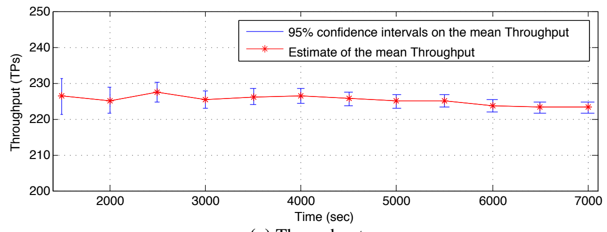
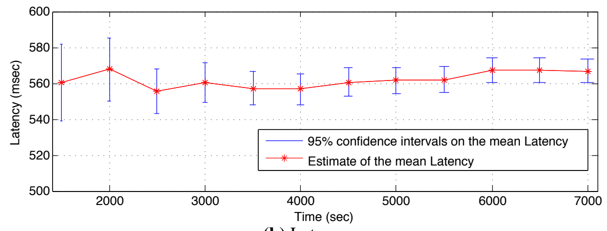


Figure 11: Variance in the Cloud – Demonstrating the erratic service of a commercial DBaaS running TPC-C workload on multiple instances of the same types on two DBaaS's.



(a) Throughput



(b) Latency

Figure 12: Variance in the Cloud – Performance measurements over an extended period of time.

on a large instance of a DBaaS and we computed the cumulative mean throughput and latency. The results in Fig. 12 show that up to a certain point, the longer we run the benchmark the tighter becomes the 95% confidence interval on both throughput and latency. This means that over time we can collect more accurate conclusions on the instance's performance. Similarly, by testing additional instances of the same type and from the same data center, we obtain confidence intervals on the data center's quality of service.

7.2 Performance-vs-Cost Comparison

Next, we used OLTP-Bench to measure the performance-vs-cost ratio of a single DBaaS provider. Such a comparison allows a user to decide what the right trade-off is between performance and cost for a given application. Using the YCSB and Wikipedia benchmarks, we used OLTP-Bench to deploy databases on five different instance sizes on Amazon's RDS platform (Table 3). We then ran each benchmark separately at its maximum speed for a total of 30 minutes. During that time, OLTP-Bench calculates the average maximum sustained throughput and the 95th percentile latency from the middle 20 minutes of the experiment's run.

The graph in Fig. 13 shows these 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 that price point does not yield any significant throughput improvement. We suspect that this because of two possible causes. First, since the more expensive instances might be co-located with other busy instances, there is increased resource contention. Another possibility is that the throughput plateaus because the disk becomes the main bottleneck, though we

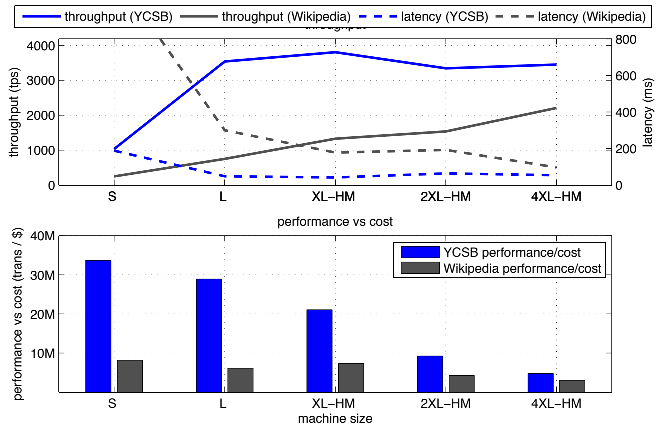


Figure 13: DBaaS Performance-vs-Cost – Comparing RDS using the Wikipedia and YCSB workloads within same data center.

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 3: EC2 RDS Experimental Systems

can only speculate on that point since this is an OS statistic that OLTP-Bench is unable to retrieve from a DBaaS.

Fig. 13 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 as 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 outweighs their performance advantage.

7.3 Performance-vs-Cost DBMS Comparison

Similarly to the previous experiment, we now compare the performance/cost trade-off of the two different DBaaS offerings available from Amazon RDS. We use YCSB in this evaluation because of its diverse transactions. We again use OLTP-Bench to deploy the benchmark's database on five different instance sizes (Table 3) and measure their throughput and latency over a 30 minute period.

The results in Fig. 14 show that the two underlying DBMSs in RDS behave similarly with the only exception of an unexpectedly lower latency for DBMS-W. There are several interesting patterns that emerge from the data collected by OLTP-Bench. Foremost is that the performance/cost ratios for both systems get worse when running on bigger instances. The best choice for both DBMS-W and DBMS-K appears to be once again the L instance.

These results show that performance and cost are complex aspects in a DBaaS context, and that it is hard to quantify such metrics without a testbed like ours.

7.4 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

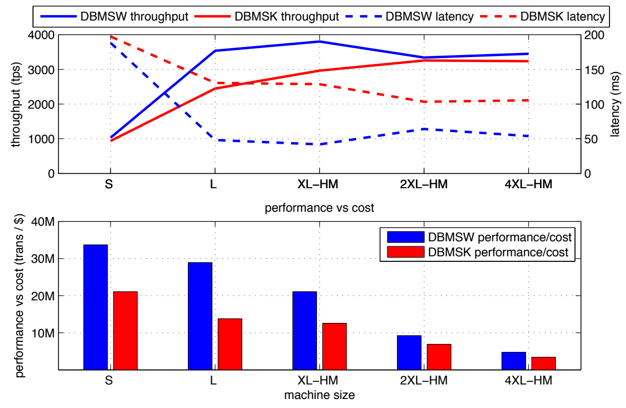
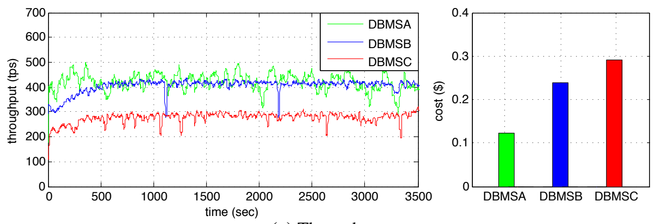
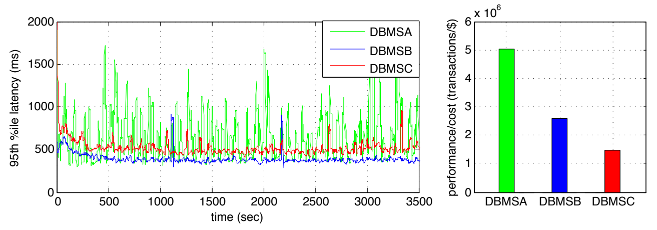


Figure 14: DBaaS Performance-vs-Cost DBMS Comparison – Comparing the performance of YCSB running on two DBMS back-ends on Amazon's RDS platform.



(a) Throughput



(b) Latency

Figure 15: Comparing DBaaS Providers: Performance trade-off across RDS offerings and SQL Azure for SEATS across data-centers (New opaque names and different workload to preserve anonymity of vendors.)

virtual machines hosted in the same data center region as the managed instances to ensure fairness between the DBaaS providers. The results in Fig. 15 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 these two aspects of cloud-based deployments.

7.5 Discussion

Performance and cost are just two of the many dimensions that one must consider when choosing a database service provider. In this context, Kossmann et al. [21] conducted an end-to-end study on performance and cost using TPC-W and reported several findings, among which that service providers prioritise different aspects depending on their business model. Similary, we found that the performance/cost trade-off was the most difficult to explore and requires the use of appropriate benchmark corresponding to the target application. Although some of our findings were surprising, they coincide with anecdotal evidence from DBMS researchers in both industry and academia. This suggests that understanding how to best leverage DBaaS solutions is still a major challenge today. We believe that this is due to two factors: (1) the pricing schemes currently used by DBaaS vendors make it difficult to predict the final costs since they are typically based on complex metrics involving storage, bandwidth, and I/O, and (2) the information needed to determine the performance that one can obtain from a particular deployment is often hidden because of the lack of guarantees on raw resources. Based on our own experience in working with OLTP-Bench, this dearth of information only allows one to speculate on their cause. We believe, however, that this is likely to change as the DBaaS landscape matures, but for now benchmarking is the best line of defense for early adopters of this new generation of services.

8. RELATED WORK

Benchmarking and performance analysis received considerable attention in the late 1980s and early 1990s [10, 20, 19], when many standard synthetic benchmarks were created [36, 40, 4]. More recently, the emergence of new application domains has led to a proliferation of new benchmark specifications that target specific subfields, such as XML data stores [31], streaming data [7], key-value stores [16], table stores [24], hybrid OLAP/OLTP databases [18], dynamic websites [28], P2P systems [22], spatial databases [27], and cloud services [21]. Other benchmarking frameworks only support just one particular DBMS platform [5, 3, 37]. To the best of our knowledge, OLTP-Bench is the first database benchmarking testbed that supports both multiple DBMSs and workloads.

When designing our testbed, we embraced the principles introduced in [33], by providing tools to control the mixture and rate of the transactions and by supporting "trace-based" benchmarks based on real data. The set of workloads that we present in this paper spans several important application domains, and enables users of OLTP-Bench to easily design benchmarks that emulate real applications or stress-testing individual system features.

We strived to offer an automated benchmark controller that is easy to run and extend. Other multi-workload benchmarking frameworks, such as Shore-Kits [37], are difficult to extend and only work for a single DBMS. In that sense, the project in [34] is similar to ours, but is designed for a broader class of use cases (generic server benchmarking), focuses on a few metrics only (e.g., peak-rate), and does not provide DBMS-specific benchmarks or tools. Both OLTP-Bench and the system described in [34] can be used to help administrators tune the configuration parameters of a system by running semi-automated experiments, following a new paradigm called *experiment-driven management* that suggests to replace analytical modeling by semi-automated experiments to manage database systems [9] or cloud infrastructures [41].

9. FUTURE WORK

Although our testbed is fully functional, we hope that it is just the starting point of a long-running effort. Our current and future development plans include:

- **Native NoSQL systems support:** We plan on extending our API to support workloads on NoSQL systems natively.
- **Improved SQL Dialect Support:** We intend to enhance the SQL-Dialect Manager's ability to automatically translate each benchmark query into different SQL dialects.
- **Automatic Request Distribution Collection:** We would like OLTP-Bench to be able to automatically extract important access distribution information from real-world workloads and data and to automatically generate a benchmark that is based on this information.
- **Stored Procedures:** We plan to extend OLTP-Bench to include support for automatically generating stored procedures for each benchmark. This is particularly challenging due to the lack of

a common language to express stored procedures in the various DBMSs we support.

We also plan to leverage OLTP-Bench's growing user base to improve and extend our testbed, including adding both new OLTP and OLAP benchmarks and configuration rules that are tailored to different DBMSs and cloud environments. We hope to involve not only researchers but also application developers in this effort.

10. CONCLUSIONS

This work presented OLTP-Bench, a "batteries included" testbed for benchmarking DBMSs. OLTP-Bench allows a user to run performance analysis experiments with tight control on the mixture, rate, and access distribution of transactions for its 15 built-in workloads. The workloads include synthetic micro-benchmarks, popular OLTP benchmarks, and real-world Web applications. We conducted extensive testing that demonstrate that OLTP-Bench can be used for different types of experimental analyses on four different DBMSs and three cloud-based database services. The source code for OLTP-Bench, as well as detailed configuration parameters, raw results, and detailed graphs for all of the experiments we present in this paper are available on the project's website [2].

11. ACKNOWLEDGMENTS

The authors thank Evan Jones and Yang Zhang for initial help, Dimitri Vorona for porting benchmarks, Sam Madden, Ben Reed, Rusty Sears, Barzan Mozafari, and Ippokratis Pandis for their feedback. This work was supported (in part) by Amazon and the Swiss National Science Foundation (#PP00P2-128459).

12. REFERENCES

[1] JPA Performance Benchmark. http://www.jpab.org.
[2] OLTPBenchmark.com. http://oltpbenchmark.com.
[3] pgbench. http://postgresql.org/docs/9.2/static/pgbench.html.
[4] PolePosition: The Open Source Database Benchmark. http://polepos.org.
[5] SysBench: A System Performance Benchmark. http://sysbench.sourceforge.net.
[6] V. Angkanawaraphan and A. Pavlo. AuctionMark: A Benchmark for High-Performance OLTP Systems. http://hstore.cs.brown.edu/projects/auctionmark.
[7] A. Arasu, M. Cherniack, E. F. Galvez, D. Maier, A. Maskey, E. Ryvkina, M. Stonebraker, and R. Tibbetts. Linear road: A stream data management benchmark. In *VLDB*, 2004.
[8] T. G. Armstrong, V. Ponnekanti, D. Borthakur, and M. Callaghan. Linkbench: a database benchmark based on the facebook social graph. In *SIGMOD Conference*, pages 1185–1196, 2013.
[9] S. Babu, N. Borisov, S. Duan, H. Herodotou, and V. Thummala. Automated experiment-driven management of (database) systems. In *HotOS*, 2009.
[10] D. Bitton, D. J. DeWitt, and C. Turbyfill. Benchmarking database systems a systematic approach. In *VLDB*, 1983.
[11] M. J. Cahill, U. Röhm, and A. D. Fekete. Serializable isolation for snapshot databases. SIGMOD, pages 729–738, 2008.
[12] M. J. Cahill, U. Röhm, and A. D. Fekete. Serializable isolation for snapshot databases. *ACM Transactions on Database Systems (TODS)*, 34(4):20, 2009.
[13] R. Cattell. Scalable SQL and NoSQL data stores. *SIGMOD Rec.*, 39:12–27, 2011.
[14] M. Cha, H. Haddadi, F. Benevenuto, and K. P. Gummadi. Measuring user influence in Twitter: The million follower fallacy. In *ICWSM*, May 2010.
[15] R. Cole, F. Funke, L. Giakoumakis, W. Guy, A. Kemper, S. Krompass, H. Kuno, R. Nambiar, T. Neumann, M. Poess, et al. The mixed workload ch-benchmark. In *Proceedings of the Fourth International Workshop on Testing Database Systems*, page 8. ACM, 2011.
[16] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with ycsb. In *SoCC*, pages 143–154, 2010.
[17] 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.
[18] F. Funke, A. Kemper, and T. Neumann. Benchmarking hybrid OLTP & OLAP database systems. In *BTW*, pages 390–409, 2011.
[19] J. Gray. *Benchmark Handbook: For Database and Transaction Processing Systems*. Morgan Kaufmann Publishers Inc., 1992.
[20] W. H. Highleyman. *Performance Analysis of Transaction Processing Systems*. Prentice Hall, 1989.
[21] D. Kossmann, T. Kraska, and S. Loesing. An evaluation of alternative architectures for transaction processing in the cloud. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 579–590. ACM, 2010.
[22] M. Lehn, T. Triebel, C. Gross, D. Stingl, K. Saller, W. Effelsberg, A. Kovacevic, and R. Steinmetz. Designing benchmarks for p2p systems. In *From Active Data Management to Event-Based Systems and More*. 2010.
[23] P. Massa and P. Avesani. Controversial users demand local trust metrics: an experimental study on epinions.com community. In *AAAI-05*, pages 121–126, 2005.
[24] S. Patil, M. Polte, K. Ren, W. Tantisiriroj, L. Xiao, J. López, G. Gibson, A. Fuchs, and B. Rinaldi. YCSB++: benchmarking and performance debugging advanced features in scalable table stores. SOCC, pages 9:1–9:14, 2011.
[25] 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.
[26] D. R. Ports and K. Grittner. Serializable snapshot isolation in postgresql. *Proceedings of the VLDB Endowment*, 5(12):1850–1861, 2012.
[27] S. Ray, B. Simion, and A. Brown. Jackpine: A benchmark to evaluate spatial database performance. In *ICDE*, 2011.
[28] E. Sarhan, A. Ghalwash, and M. Khafagy. Specification and implementation of dynamic web site benchmark in telecommunication area. In *WEAS*, pages 863–867, 2008.
[29] Scalyr. Even Stranger than Expected: a Systematic Look at EC2 I/O. http://blog.scalyr.com/2012/10/16/a-systematic-look-at-ec2-io/.
[30] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz. Runtime measurements in the cloud: Observing, analyzing, and reducing variance. *PVLDB*, 3(1), 2010.
[31] A. Schmidt, F. Waas, M. Kersten, M. J. Carey, I. Manolescu, and R. Busse. Xmark: a benchmark for xml data management. In *VLDB*, 2002.
[32] B. Schroeder, A. Wierman, and M. Harchol-Balter. Open versus closed: a cautionary tale. NSDI, pages 18–18, 2006.
[33] M. Seltzer, D. Krinsky, K. Smith, and X. Zhang. The case for application-specific benchmarking. In *HotOS*, 1999.
[34] P. Shivam, V. Marupadi, J. Chase, T. Subramaniam, and S. Babu. Cutting corners: workbench automation for server benchmarking. In *USENIX*, 2008.
[35] M. Stonebraker and A. Pavlo. The SEATS Airline Ticketing Systems Benchmark. http://hstore.cs.brown.edu/projects/seats.
[36] The Transaction Processing Council. TPC-C Benchmark (Revision 5.9.0). http://www.tpc.org/tpcc/spec/tpcc_current.pdf, June 2007.
[37] P. Tözün, I. Pandis, C. Kaynak, D. Jevdjic, and A. Ailamaki. From A to E: analyzing TPC's OLTP benchmarks: the obsolete, the ubiquitous, the unexplored. EDBT, pages 17–28, 2013.
[38] G. Urdaneta, G. Pierre, and M. van Steen. Wikipedia workload analysis for decentralized hosting. *Comput. Netw.*, 53:1830–1845, July 2009.
[39] G. Weikum. Where is the Data in the Big Data Wave? http://wp.sigmod.org/?p=786.
[40] A. Wolski. TATP Benchmark Description (Version 1.0). http://tatpbenchmark.sourceforge.net, March 2009.
[41] W. Zheng, R. Bianchini, G. J. Janakiraman, J. R. Santos, and Y. Turner. Justrunit: experiment-based management of virtualized data centers. In *USENIX*, 2009.