BenchPress: Dynamic Workload Control in the OLTP-Bench Testbed

Dana Van Aken
Carnegie Mellon University
dvanaken@cs.cmu.edu

Djellel E. Difallah
University of Fribourg
djelleleddine.difallah@unifr.ch

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

Carlo Curino
Microsoft Corporation
ccurino@microsoft.com

Philippe Cudré-Mauroux
University of Fribourg
philippe.cudre-mauroux@unifr.ch

ABSTRACT

Benchmarking is an essential activity when choosing database products, tuning systems, and understanding the trade-offs of the underlying engines. But the workloads available for this effort are often restrictive and non-representative of the ever changing requirements of the modern database applications. We recently introduced OLTP-Bench, an extensible testbed for benchmarking relational databases that is bundled with 15 workloads. The key features that set this framework apart is its ability to tightly control the request rate and dynamically change the transaction mixture. This allows an administrator to compose complex execution targets that recreate real system loads, and opens the doors to new research directions involving tuning for special execution patterns and multi-tenancy. In this demonstration, we highlight OLTP-Bench's important features through the BenchPress game. It allows users to control the benchmark behavior in real time for multiple database management systems.

Categories and Subject Descriptors

H.3.4 [**Systems and Software**]: Performance evaluation

General Terms

Experimentation, Performance

Keywords

Benchmarking, Configuration, Tuning

1. INTRODUCTION

New database initiatives are motivated by either emerging use-cases or the need to improve existing deployments. For these efforts to be successful, it is important to use precise and flexible measurement tools for comparing database management systems (DBMSs) and stressing them under different circumstances. One such way is use benchmarks, as it allows one to understand and compare the performance of these systems. Over the years, benchmarking has evolved from a set of simple routines that generate a single performance number to become what is now often a complex effort involving different workloads, parameters, and other variables [5].

Database administrators and researchers test DBMSs using either common industry standard benchmarks or, if need be, custom workloads [2, 6]. In the latter case, the code and the data sets (if any) for these workloads are not always available or are not well maintained. Thus, this makes it difficult for others to verify results from previous projects, or to port the benchmarks to additional DBMSs. Over the years, we noticed that many of the software components that we and other researchers built for evaluating DBMSs are reusable. Making this software available to the database community fosters and encourages experimental repeatability.

For these reasons, we developed the OLTP-Bench benchmarking testbed that is aimed at making it easier to reliably and repeatedly evaluate DBMSs [4]. OLTP-Bench is capable of dynamically controlling the transaction rate, mixture, and workload skew during the execution of an experiment. This allows one to simulate a multitude of practical scenarios that are typically hard to test (e.g., time-evolving access skew). Our framework provides an easy way to monitor the performance and resource consumption of the database system under test. It currently supports over 15 benchmarks, including synthetic micro-benchmarks, OLTP benchmarks, and real-world Web applications. These were ported by the authors of this work and as well as from several contributors in the community.

One challenging aspect that we focused on while building OLTP-Bench is the ability to control the rate of requests with great precision. As we describe in Section 2, this is hard to achieve for multiple DBMSs in a single codebase. Moreover, OLTP-Bench also supports changing transaction request rates dynamically during execution based on user-defined workloads. With these two features, one is able to design complex execution scenarios in OLTP-Bench. For example, one can run multiple workloads in parallel to test a DBMS's ability to support multi-tenant deployments.

In this demonstration, we showcase the dynamic and flexible control features of OLTP-Bench through **BenchPress**. BenchPress is a graphical interface that allows users to control OLTP-Bench's behavior in real-time. It supports the dynamic modification of a benchmark's transaction workload mixture and throughput rates, as well as the execution of additional benchmarks on-the-fly. The demonstration also allows users to compare different DBMSs within the same framework.

We next provide an overview of the key technical contributions of OLTP-Bench in Section 2, and discuss the datasets and benchmarks available in our demo in Section 3. Finally, we discuss in Section 4 how BenchPress allows the user to play with these various aspects in our testbed for the demo.

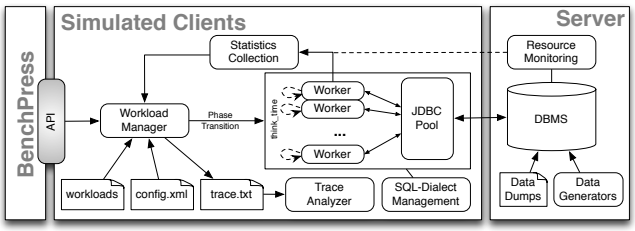


Figure 1: OLTP-Bench Architecture – The client-side handles workers and generates the transaction workload according to a configuration provided by the user, or via the real-time control API. On the left side, BenchPress utilizes the API to send the commands and to track the execution in real time. The framework also employs monitoring tools to gather server-side resource utilization statistics.

2. OVERVIEW

OLTP-Bench is an extensible, "batteries included" database benchmarking testbed [4]. It works with a number of single-node DBMSs, distributed DBMSs, and DBaaS systems that supports SQL through JDBC. As shown in Fig. 1, the architecture of our framework is comprised of two main components: (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 client-side portion is small and portable (less than 5MB). The framework has been tested and deployed on a variety of Unix-like platforms.

2.1 Architecture

OLTP-Bench's client-side component contains a centralized *Workload Manager* that is responsible for tightly controlling the characteristics of the workload via a centralized request queue. It takes as input a configuration file describing a predefined workload with multiple execution phases, where a phase is defined as (1) a target transaction rate, (2) a transaction mixture, and (3) a time duration in seconds.

The Workload Manager spawns multiple client *Worker* threads that each connect to the target DBMS using JDBC and iteratively pull tasks from the request queue. For each new transaction request, a Worker invokes the corresponding transaction's control code (i.e., program logic with parameterized queries) and either commits or aborts the transaction.

To handle portability across multiple DBMS SQL dialects, we decided to use support human-written dialect translation instead of automatic tools. In that way, we allow experts for individual systems to contribute specific SQL variants—both for DML and DDL queries and operations—for different systems.

On the server side, we use standard server monitoring tools [8] that are launched in parallel to OLTP-Bench and provide system performance metrics in real time as they are collected on the host.

2.2 Features

In [4], we introduced the requirements that motivated the design decisions behind OLTP-Bench. We provide below an overview of these key features that we implemented.

2.2.1 Rate Control

The ability to control request rates with great precision in a DBMS is important for understanding performance anomalies. Even small oscillations in the throughput can make the interpretation of results difficult. OLTP-Bench can either execute transactions in an open loop fashion or with a throttled transaction per second rate for predefined periods of time [7]. This allows one to evaluate how well a DBMS can sustain long periods of continuous load.

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.

As described above, the runtime throughput is controlled through the Workload Manager's request queue. At runtime, the manager generates new requests and add them to this queue. The Workers pull a request from the queue, execute it, sleep for an optional "think time" period, and then return the queue for a new request. Using a centralized queue allows us to control the throughput from one location without needing to coordinate the multiple Worker threads. The exact number of requests configured is added to the queue each second, and each arrival is interleaved with a uniform or exponential arrival time. When the workers cannot keep up with all requests, the remainder is postponed in such a way that the framework never exceeds the target rate. In case an unlimited throughput is requested, the arrival is set to a large configurable constant.

2.2.2 Mixture Control

While the Workload Manager inserts work requests into the queue, the workers choose the benchmark's specific transactions to execute by sampling from a predefined distribution (or mixture). In OLTP-Bench, we added the ability to change the mixture of transactions used in a given benchmark in every phase, or on demand via the API. This allows the user to experiment with different combinations [4], for example by transitioning from read-heavy to write-heavy workloads.

2.2.3 Multi-tenancy

OLTP-Bench can be configured to run multiple workloads and benchmarks in parallel. A novel feature that we introduce allows the users to perform multi-tenancy tests that isolate different workloads within the same instance. This is especially important in Database-as-a-Service (DBaaS) deployments [3].

For the purpose of this demo and in response to user feedback, we created a application programming interface (API) for OLTP-Bench that exposes the ability to programmatically control its execution at the runtime. In addition, this API also provides instantaneous feedback about the current execution throughput and average latency per transaction type. As we discuss in Section 4, this API enables us to turn our benchmark system into an interactive game. More importantly, an API for controlling the execution load facilitates the integration of our system in the context of broader test infrastructures. This could be useful to dynamically create new workload mixtures in response to application-level observations.

3. BENCHMARK DATA & WORKLOADS

The recent growth in Web and mobile-based applications requiring transactional support pushed the boundaries of traditional benchmarks. Instead of trying to be exhaustive, we chose an initial

set of benchmarks that covers a number of currently popular applications. Table 1 gives an overview of the 15 benchmarks currently ported to OLTP-Bench, along with their application domain. We believe that each benchmark in that table is useful in modeling a specific application domain. We note that the size of the database corresponding to each benchmark is configurable by the administrator and that the working set size can be automatically scaled. More detailed information, including descriptions of the individual transactions in each benchmark, is available on our website [1].

4. DEMONSTRATION DESCRIPTION

BenchPress is a game that allows users to control the behavior of OLTP-Bench through its API. The objective is to be able to navigate a game character throughout a scrolling obstacle course. The vertical height of the character at a given point in time is based on the current throughput (transactions per second) of the target DBMS. The user controls their character by increasing or decreasing the target throughput using the keyboard or the controller. The character, however, only responds to the actual throughput delivered by the DBMS as measured by OLTP-Bench. The boundaries of obstacles correspond to different target throughput rates. If the DBMS cannot deliver the requested transaction rate, then the character will crash into an obstacle.

BenchPress is a JavaScript application that runs in a browser. It connects to a Web-based application server that that connects to OLTP-Bench. The benchmark framework is deployed on a machine that contains several target DBMSs.

Our system features many tests to challenge the user, for example by linearly increasing or decreasing the execution patterns using higher and lower obstacles. This demonstration allows users to (1) gain insight about the benchmarks included in OLTP-Bench, (2) become familiar with the functionalities offered by OLTP-Bench, and (3) stir a discussion on how a specific execution pattern might influence the performance of a DBMS and potentially expose hidden weaknesses.

4.1 BenchPress Gameplay

Our demo is a side-scrolling game where the character is indirectly controlled using a keyboard or an external input device. As shown in Fig. 2a, the user starts the game by picking the desired benchmark. Each benchmark corresponds to a different character in the game. They then select the target DBMS (Fig. 2b). Each DBMS corresponds to a different stage with varying environment conditions. For example, the screenshot in Fig. 2c shows that MySQL is the forest level.

The character has to progress through a series of *obstacles* by either *jumping* over them or letting the character *fall* due to the simulated gravity. We define the previous concepts as follows:

- An *obstacle* is a set of vertical "pipes" that limits the character's movement within a defined range. This range is given by the height of the pipes and represents the expected throughput at which the challenge is preset for a given period of time.
- A *jump* requests a higher throughput rate, hence it makes the game character move upwards. The movement of the character however only reflects the actual throughput delivered by the DBMS rather than the requested one. This measures the responsiveness of the DBMS to changes in the OLTP-Bench's requested load.
- A *fall* makes the game character go down following some simulated gravity, in the sense that the throughput automatically decreases linearly until reaching 0 transactions per second, at which point the character falls on the floor. A different setup would allow the user to manually decrease the throughput using the commands.

4.1.1 Mixture Control

In addition to the basic controls described above, the user can alter the benchmark mixture on-the-fly. That is, the user can pause BenchPress at any moment in time to change the workload parameters order to avoid an obstacle. Beside the ability to fully customize a workload by manually assigning new probability distributions for the transactions, BenchPress includes preset mixtures. As shown in Fig. 2d, these include "read-heavy" and "write-heavy" workload mixtures. Modifying the workload mixture allows for example players to have a tighter control on the character (effectively on the throughput) when the DBMS struggles at maintaining the rate required to pass some difficult obstacle.

4.1.2 Challenges

Our goal is to create a simulated load that the DBMS must respond to. To that end, the challenges represent the throughput to achieve during the game at any given point in time. In BenchPress, challenges take the form of obstacles with a narrow opening, fixing the expected throughput range to achieve visually.

Other challenges in the game are *auto-pilot zones*, where the user has to identify the right throughput and mixture that allows the character to pass through the obstacles successfully without any external input. That is, they are not able to control the throughput as their game character moves through these zones. In that case, the obstacle is a target throughput that has to be achieved for a given period of time. This challenge will make the user reflect on the different parameters that can be used to reach a given target execution.

For this demo, we created challenges following four different shapes (although this list is not exhaustive, new challenges can be created using a configuration file):

Steps: The character has to go through a set of increasing or decreasing throughput levels. This simulates an increasing load on the database; at some point the DBMS will become saturated and be unable to process any more transactions. In the worst case, the performance may actually get worse depending on the workload.

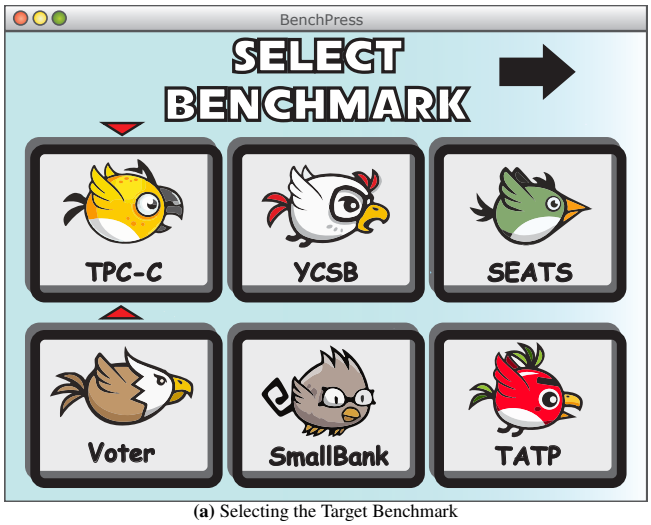
Sinusoidal: The character has to move up and down in a recurring pattern. This demonstrates a fluctuating load and tests the ability of the DBMS to gracefully respond without much jitter.

Peak: After a period of low throughput simulating some steady-state workload, a peak in throughput is created for a short period before going back to normal. Again, this will show the ability of a DBMS to respond to some sporadic and sudden increase in load.

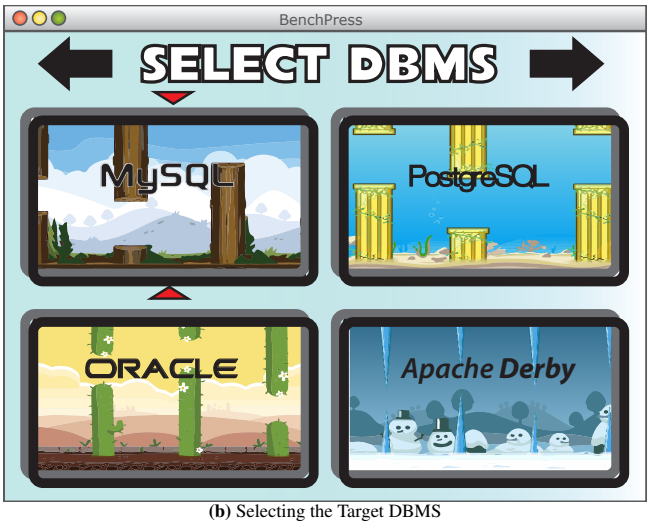
Tunnels: The auto pilot zones are long tunnels where the target execution is fixed to a constant range of high (or low) target throughput. This challenge expects the DBMS to deliver a constant tight throughput for a long period of time.

4.2 Performance Visualization

The BenchPress interface provides a visual overview about the DBMS's performance in terms of throughput and latency. To complement this information, the OLTP-Bench monitoring tool will display in real time the metrics collected from the system on which the DBMS is running. This information can be useful for the user to predict potential drops in performance e.g., when getting close to being CPU-bound. Hence, the user can take the necessary actions to prevent an eventual crash into an obstacle by tuning down the transaction rate and potentially causing a performance drop (see



(a) Selecting the Target Benchmark



(b) Selecting the Target DBMS

(c) Main Game Screen

(d) Dynamically Change the Workload Mixture

Figure 2: The BenchPress game screenshots.

Section 4.1.1). For example, the user could in that context lower the write-intensive transactions if the disk IO activity seems to saturate.

4.3 Demo Takeways

The goals of this demo are threefold. First, we aim at engaging the audience with an interactive demonstration that goes beyond the typical back-end demonstrations of DBMSs. Second, we seek to showcase OLTP-Bench's ability to control a multiplicity of database benchmarking parameters dynamically. Lastly, we hope that the game provides users with a number of key insights about DBMSs and transactional workloads. Examples of this include understanding a DBMS weaknesses and the idiosyncrasies of the various workloads that are built into OLTP-Bench (cf. Table 1). The player will learn that certain types of transactions are more difficult to sustain than others, that some cannot be used to achieve high throughput, or that certain DBMSs (and tuning combinations) cannot pass the tunnel tests, since they produce oscillating throughputs. Moreover, the two-player version of the game allows the players to experience in real-time the effects of multi-tenancy, with one player affecting the other.

5. REFERENCES

[1] OLTPBenchmark.com. http://oltpbenchmark.com.

[2] 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.

[3] C. A. Curino, D. E. Difallah, A. Pavlo, and P. Cudre-Mauroux. Benchmarking oltp/web databases in the cloud: the oltp-bench framework. In *Proceedings of the fourth international workshop on Cloud data management*, pages 17–20. ACM, 2012.

[4] D. E. Difallah, A. Pavlo, C. Curino, and P. Cudré-Mauroux. OLTP-Bench: An Extensible Testbed for Benchmarking Relational Databases. *PVLDB*, 7(4):277–288, 2013.

[5] J. Gray. *Benchmark Handbook: For Database and Transaction Processing Systems*. Morgan Kaufmann Publishers Inc., 1992.

[6] 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.

[7] B. Schroeder, A. Wierman, and M. Harchol-Balter. Open versus closed: a cautionary tale. NSDI, pages 18–18, 2006.

[8] D. Wieers. Dstat: Versatile resource statistics tool. http://dag.wiee.rs/home-made/dstat.