HTAPBench: Hybrid Transactional and Analytical Processing Benchmark

Fábio Coelho[*]
INESC TEC & U.Minho
Braga, Portugal
fabio.a.coelho@inesctec.pt

João Paulo[†]
INESC TEC & U.Minho
Braga, Portugal
jtpaulo@di.uminho.pt

Ricardo Vilaça
LeanXcale SL
Madrid, Spain
ricardo.vilaca@leanxcale.com

José Pereira
INESC TEC & U.Minho
Braga, Portugal
jop@di.uminho.pt

Rui Oliveira
INESC TEC & U.Minho
Braga, Portugal
rco@di.uminho.pt

ABSTRACT

The increasing demand for real-time analytics requires the fusion of Transactional (OLTP) and Analytical (OLAP) systems, eschewing ETL processes and introducing a plethora of proposals for the so-called Hybrid Analytical and Transactional Processing (HTAP) systems.

Unfortunately, current benchmarking approaches are not able to comprehensively produce a unified metric from the assessment of an HTAP system. The evaluation of both engine types is done separately, leading to the use of disjoint sets of benchmarks such as TPC-C or TPC-H.

In this paper we propose a new benchmark, HTAPBench, providing a unified metric for HTAP systems geared toward the execution of constantly increasing OLAP requests limited by an admissible impact on OLTP performance. To achieve this, a load balancer within HTAPBench regulates the coexistence of OLTP and OLAP workloads, proposing a method for the generation of both new data and requests, so that OLAP requests over freshly modified data are comparable across runs.

We demonstrate the merit of our approach by validating it with different types of systems: OLTP, OLAP and HTAP; showing that the benchmark is able to highlight the differences between them, while producing queries with comparable complexity across experiments with negligible variability.

CCS Concepts

•Information systems → Database performance evaluation; Relational parallel and distributed DBMSs;

[*]Funded by source 1 (Acknowledgments).

[†]Funded by source 2 (Acknowledgments).

ICPE'17, April 22-26, 2017, L'Aquila, Italy
© 2017 ACM. ISBN 978-1-4503-4404-3/17/04...$15.00
DOI: http://dx.doi.org/10.1145/3030207.3030228

Keywords

Benchmarking; OLTP; OLAP; HTAP

1. INTRODUCTION

We are undergoing a change in database system design. Up until now, there has clearly been two main categories into which systems could be classified: operational or transactional systems (OLTP) and data warehousing or analytical systems (OLAP). For a long time, each system type lived apart from each other, as each had very distinct goals. On the one hand, transactional systems focus their operation on providing ways to enable a high throughput of rather small-sized transactions. Typically, during execution, each transaction operates under a very limited space of tuples. OLTP systems use a data schema based on reduced levels of redundancy, resulting from normalization techniques [26] across entities. On the other hand, analytical systems focus on performing aggregations over large sets of data. The query centric nature of an OLAP data schema introduces high levels of redundancy in the shape of materialized views or precomputed aggregates [19].

As pointed out by Gartner [28], a new class of engines, capable of handling mixed workloads with high levels of transactional activity and, at the same time, providing scalable business analytics directly over production data is arising. Such systems bypass the need to use the ETL [18] process and are commonly referred to as HTAP - Hybrid Analytical and Transactional Processing systems. Traditionally, both workloads are handled by separate engines, periodically feeding the OLAP with data from the OLTP engine through an ETL process. The approach seeks to ensure the best performance of each individual engine at the expense of data freshness for analytics. By eschewing the ETL process, HTAP systems are poised to reduce implementation, management and storage costs and, most importantly, enable real-time analytics over production data. Several vendors such as Oracle, SAP or PostgreSQL started to fill in this gap and proposed solutions that coined variant terminologies such as OLTAP [21](Online Transactional Analytical Processing) or OLxP [16, 22] (Online Transctional or Analytical Processing) that also aim to mix both workload types. Other companies have started to offer Hybrid solutions such as NuoDB [24], VoltDB [27] or Splice Machine [20].

Given the traditional taxonomy of OLTP and OLAP systems, the industry along with independent organizations defined benchmarking approaches specially tailored for either transactional or analytical workloads, such as TPC-C [8] and TPC-E [11] for OLTP workloads and TPC-H [9] or TPC-DS [23] for analytical workloads. Each benchmark focuses on the optimization challenges associated with each system type, defining evaluation suites with very different and contradicting goals. This is so as optimizing an OLTP targeted operation would intrinsically degrade OLAP performance and vice-versa [14]. For instance, OLTP and OLAP workloads generate specific sets of queries that require distinct storage layouts in order to be efficient. They can also accommodate different-sized datasets, employing the concept of warehouse as scaling factor. Optimizing a storage layout to support both access patterns efficiently is not a trivial task but it must be accomplished to have efficient hybrid systems. Moreover, as further discussed in the paper, storage accesses generated by OLTP workloads are mostly random while OLAP workloads are mainly sequential. Likewise, a hybrid workload will assess the ability of the system under test (SUT) to simultaneously schedule random and sequential access patterns to storage mediums and manage both light and intense operations regarding memory allocation and processor time; these are some of the reasons why these workloads have until now been evaluated independently.

Most importantly, it is not easy to combine and directly translate results from distinct benchmarks to the effectiveness of a system to handle an HTAP workload. Gartner states that an HTAP system should prioritize a sustained transactional throughput, delivering at the same time scalable analytical processing without disrupting the operational activity [28]. Consequently, even if both workloads can be run on the same engine, it is not straightforward to meaningfully and consistently reconcile the results of both workloads in a single HTAP metric. This is so as each workload is usually oblivious to the presence of the other, trying to independently reach the maximum qualified throughput in each separate workload, and therefore producing uncorrelated metrics.

In this paper, we present HTAPBench, a new benchmark suite designed to evaluate hybrid systems with mixed OLTP and OLAP workloads. HTAPBench introduces a new metric that provides a reading of the analytical capability as the system scales. The proposed benchmark introduces a hybrid workload that simultaneously exercises a workload with operational and analytical activity over the same system. It introduces a Client Balancer that controls how analytical clients are launched, ensuring that the OLTP activity stays within a configured threshold and the results are kept comparable across runs by addressing data uniformity of the workload. Throughout the paper we use *database* when referring to the stored data, *engine* when referring to the software and *SUT* or *system* when referring to the composition of software and underlying hardware.

Contributions: First, we propose a new unified metric aimed at engines with mixed workloads. Second, we present the design and implementation of HTAPBench supporting such a metric. We introduce a technique for generating realistic OLAP queries over a dynamically changing dataset resulting from concurrent OLTP, with predictable complexity. In addition, we propose a load balancer that relies on a feedback control mechanism that regulates the coexistence of the OLTP and OLAP workload execution.

Roadmap: The remainder of this paper is organized as follows: Section 2 presents an overview and the unified metric we propose. Section 3 evaluates our proposal against different systems, and Section 4 validates the results as well as discusses the properties of the system. Section 5 goes through related work and Section 6 concludes this work.

2. HTAPBench

The Hybrid Transactional and Analytical Processing Benchmark is targeted at assessing engines capable of delivering mixed workloads composed of OLTP transactions and OLAP business queries without resorting to ETL. Typically, in environments with mixed workloads, the relative weight given to OLTP and OLAP is governed by delivering a high OLTP throughput while still being able to simultaneously perform analytical business queries [28]. This goal should be met in such a way that the OLTP throughput is kept within expected intervals. Likewise, HTAPBench focuses its operation on ensuring a stable OLTP throughput and assessing the capability of the SUT to cope with an increasing demand on the OLAP counterpart. To fulfill OLTP requirements, HTAPBench requires the testing engine to grant full ACID capabilities during all stages of the benchmark. The design is composed of several modules as depicted in Figure 1. The Density Consultant, the Client Balancer and the Dynamic Query-H Generator modules provide the foundation of our approach and are discussed in this Section.

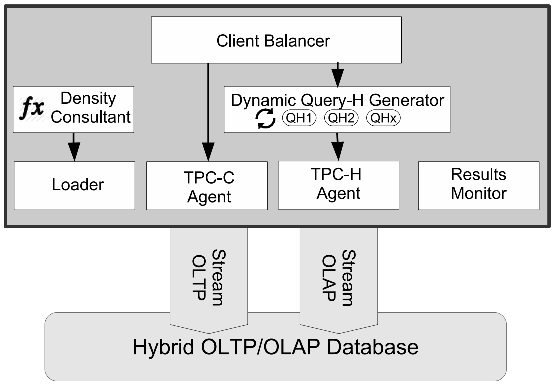


Figure 1: HTAPBench architecture.

In a nutshell, HTAPBench decomposes the execution into three main stages: (*i*) the populate stage, (*ii*) the warmup stage and (*iii*) the execution stage. Two of the modules, which we define as agents, will regulate the OLTP and OLAP activity. During system start, HTAPBench will be configured with a target OLTP throughput, triggering an OLTP workload configured with the required number of clients to meet the required throughput. Periodically, HTAPBench

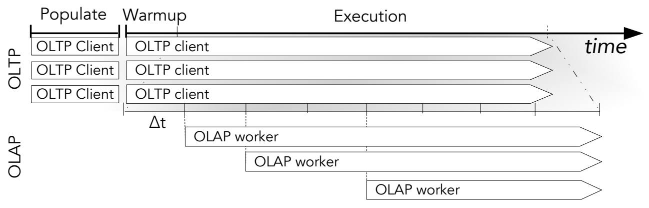


Figure 2: HTAPBench execution.

will assess the ability of the SUT to handle an increasing OLAP activity, as depicted in Figure 2, while ensuring that the transactional throughput does not decrease below a configured threshold.

The unified metric is central to our design, and its genesis is directly translated from the need of HTAP systems to scale without disturbing the OLTP activity. It mirrors the ability of a given analytical worker to complete queries, in a scenario composed of an increasing number of analytical workers and a stable transactional activity. Analyzing this behavior will enable us to identify situations where adding an additional OLAP worker degrades the OLTP/OLAP engine performance.

2.1 Workload

The mixed workload used in this benchmark is composed of a transactional agent and an analytical agent that simultaneously instruct the system to perform operations over the same dataset. We selected TPC-C and TPC-H as agents, as each is able to stress the inherent characteristics of each workload type. TPC-C was chosen as the transactional client due to its high rate of read-write operations, being one of the most used workloads for OLTP evaluation. TPC-C specification models a real-world scenario where a company, comprised of several warehouses and districts, processes orders placed by clients. The workload scales according to the number of configured warehouses.

TPC-H also specifies a real-world scenario, modeling a wholesale supplier and employing a schema that is very close in structure to TPC-C. Moreover, we selected TPC-H over TPC-DS [10] since the workload in TPC-DS is data warehouse-driven, not only relying on a star schema but also requiring the use of ETL processes to keep data updated and in conformity with such a schema. On the basis that analytical queries in a hybrid workload should exercise a dataset common to the operational workload, TPC-H better fits the requirement as it does not use a star schema, placing it closer to the workload schema in TPC-C.

The mixed workload in HTAPBench uses all the entities in TPC-C and TPC-H's `Nation`, `Region` and `Supplier`, as proposed in [5]. The remaining TPC-H entities were merged into a non-intrusive way in TPC-C's workload. The result is a workload that matches Gartner's recommendations for hybrid workloads, where data should not be moved from operational to data warehouses in order to support analytics, but live under the same schema allowing drill-down analytical operations to point toward the freshest data produced by the operational activity.

The OLTP execution in HTAPBench runs according to a target number of transactions per second (*tps*). It is thus necessary to ensure the optimal configuration regarding some TPC-C specific parameters such as the total number of warehouses and clients defining the number of transactions per second (*tpmC*).

$$target(tpmC) = target(tps) \times 60 \times \frac{\%NewOrder}{100} \tag{1}$$

$$\#clients = \frac{target(tmpC)}{1.286} \tag{2}$$

$$\#warehouses = \frac{\#clients}{10} \tag{3}$$

To compute these parameters, we refer to the TPC-C specification [8] and use the characterization for the TPC-C's ideal client, considering the minimum think time for each transaction type, and provided that transactions do not fail and thus no rollback operation is required. According to TPC-C, a single client should not be able to execute more than 1.286 *tpmC*. Under these conditions, it is possible to extract the target *tpmC* from the target *tps* (expression 1), as well as the total number of *clients* (expression 2) and *warehouses* (expression 3). The required target *tps* is one of the configurable criteria in HTAPBench and directly relates with the expected scalability of the system and respective database size (further details are provided in Section 2.4).

The business queries in TPC-H are built from filtering, aggregation and grouping operations over a given result set. Filtering operations use SQL operators such as `where`, `having` or `between`. Their main goal is to limit the number of considered rows. Since the transactional activity will feed the analytical queries in HTAPBench, the number of rows filtered by the latter will grow over time. If not addressed, the results of these analytical queries are poised to become incomparable across runs. Data distributions regulate how the parameters for filtering operators are selected, enabling the queries to dynamically exercise several regions of the dataset while exhibiting comparable complexity. On the other hand, if the queries are not dynamically generated, the use of fixed bounds on the filters would end up traversing the full domains, preventing the query planner of the SUT to be exercised.

To verify the impact of using fixed or dynamic parameters, we conducted an experiment where we considered the execution of the 22 TPC-H queries over 2 setups. The first used a set of fixed parameters that would resemble full domain searches; the second used dynamically generated parameters without being bound to a data distribution. The configuration of dynamically generated parameters created a new set for each run, while the fixed configuration reuses the same set across runs. Each setup considers the average of 5 independent executions. Queries are computed against a column-oriented engine. In each run, the database is populated with one warehouse and the queries are executed without any of the filtering operators in their composition, establishing a baseline comparison that represents the universe of rows in each query.

The experiment observes the average difference of result set row count in consecutive executions of each run for a given setup, as a percentage of the baseline result. When using fixed parametrization, the result set cardinality did not change across consecutive runs, and in most cases, queries ended up selecting a considerably broader space of tuples. When we used dynamically generated filters in the TPC-H queries, a variation of up to 77% in result set cardinality was observed in comparison with the previous approach. This is due to not using a distribution to feed the date fields during the population stage of the benchmark. By not using a distribution to regulate how these fields are generated, it becomes likely that the items inserted during the populate stage of the benchmark present uneven time distributions when compared with the ones created during the execution stage. The next Section introduces a way to generate a workload distribution that ensures analytical queries with comparable complexity across runs.

2.2 Result Set Homogeneity

The analytical queries composing a hybrid workload are fed with data created or manipulated by the transactional agent, either during the initial populate stage, or during the transactional execution part of the hybrid workload. The engine qualifying as HTAP must operate under an isolation criterion that enables the analytical queries to observe data committed by the transactional agent at the time the analytical queries started. Likewise, a given analytical query should freely access the entire dataset, spanning from the first to the last committed transaction.

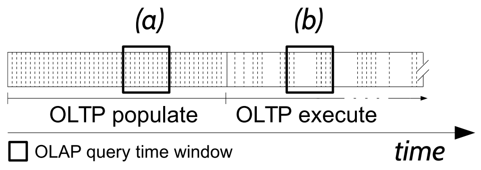


Figure 3: Timestamp density difference.

As discussed in the previous Section, the absence of a regulating mechanism would result in the use of randomized query boundaries, producing incomparable results across runs. The same may happen when analytical queries observe data generated in the populate and execution stages of the hybrid workload. Figure 3 depicts an example of the patterns that data is created or changed by the transactional agent. On the one hand, the OLTP populate stage (Figure 3a) promotes bursts of transactions inserting data, causing a high concentration of timestamps in a short time period. On the other hand, during the OTLP execution stage, the OTLP transaction rate is regulated by TPC-C.

What is desirable is that the pattern generated by the OLTP execution within TPC-C is also observed by analytical queries whenever they traverse the data loaded during the populate stage. To mitigate this issue, we introduce a density extraction mechanism that ensures the same data pattern across stages. Briefly stated, our approach observes the amount of generated date fields during the execution stage of TPC-C, allowing the system to apply the extracted density during population.

Density Function

The populate and execution stages of TPC-C generate different date densities across the whole dataset, varying according to the configured transaction mix within TPC-C. Moreover, as not all transaction types generate the same number of new timestamps, we configured our density function to reflect that behavior.

$$txnMix = \frac{\%NewOrder + \%Payment + 10 \times \%Delivery}{100} \tag{4}$$

$$d(T_S/s) = tps \times txnMix \tag{5}$$

Both the `New Order` and `Payment` transactions generate one timestamp each, while each `Delivery` transaction generates ten. The `Order Status` and the `Stock Level` transactions do not generate any timestamps. It is then possible to express density as a function between the target number of transactions per second and the ratio of `New Order`, `Payment` and `Delivery` transactions, as defined by expression 5.

In the following, we set up an experiment that allowed us to observe the expected density.

This experiment was conducted on a server with an Intel Xeon x3220 2.4 GHz QuadCore, 8GB of memory and 128GB Solid State Drive. For the purpose of this test, we relied on a Hybrid system, which we set up according to Table 1 to reflect workloads with more than 70GB in total size. In each experiment, the database was dropped and populated. Afterwards, we ran TPC-C under the standard transaction mix in runs that lasted 60 minutes.

$tpmC$	clients	warehouses
635	495	49
741	576	58
886	689	69

Table 1: Workload Configuration - Ideal TPC-C client.

The results depicted in Table 2 are the average of five independent runs regarding each target. The results depict an increasing amount of newly issued timestamps (T) as the defined target increases, thus reflecting a density function that also presents an increasing trend. The results also show

$tpmC$	Total Observed (Ts)	Expected $d(T_s/s)$	Experimental $d(T_s/s)$
635	108,051	30.24	30.01
741	125,500	35.14	34.86
886	150,114	42.02	41.69

Table 2: Density Observation Results.

that the density function provides results that are only 3% apart when comparing with the experimental observation.

The timestamp density will introduce a change in the standard TPC-C specification. It is worth noting that this modification does not introduce any change in TPC-C business logic. The individual TPC-C results are kept comparable with a same-sized TPC-C installation.

2.3 Component Design

Unified Metric

The disparity in workload complexity and structure between OLTP and OLAP workloads is a major hurdle when trying to define a unified metric for an HTAP system. So far, one of the main disadvantages of previous approaches was the fact that they would enable both OLTP and OLAP executions to grow in order to achieve the maximum throughput for each. The non-regulated growth induced by OLTP execution would inherently degrade OLAP performance since analytical queries would have to scan more data. HTAPBench removes one axis of variability by regulating the OLTP workload. The assurance of a constant transactional execution also leads to a sustained and known database growth; that is, the rate at which OLAP queries observe new data is fixed and predictable, bypassing the need to normalize the OLAP results in terms of the observed growth.

$$QpHpW = \frac{QphH}{\#OLAPworkers}@tpmC \tag{6}$$

Expression 6 defines the metric we propose, $QpHpW$ or "Queries of type H per Hour per Worker". It reads as the number of analytical queries executed per OLAP worker regarding a system that is able to sustain the configured tps.

It is defined by the ratio between TPC-H's metric and the total number of registered OLAP workers induced by the Client Balancer. A higher $QpHpW$ maps a system where each OLAP worker is able to compute more queries per analytical worker, thus representing a higher overall throughput. Achieving the best configuration for a given installation becomes a multi-run optimization problem regarding a given target tps.

Client Balancer

The Client Balancer module is responsible for monitoring and deciding whether or not to launch additional OLAP workers. When the OLTP agent ensures that the target tps is stable, the Client Balancer will periodically assess whether or not the SUT is capable of handling an extra OLAP worker. This assessment relies on a proportional-integral feedback controller.

$$output = K_P \Delta tps + K_I \int \Delta t \qquad (7)$$

The feedback control mechanism (expression 7) is characterized by a proportional (K_P) and an integral (K_I) gain adjustment. The gain parameters are used over the found system deviation ($\Delta tps = target\_tps - measured\_tps$) to compute the output value. The correct adjustment of either gain factor is vital to ensure that the feedback controller does not exceed too quickly the SUT's capabilities, which in turn would launch a higher number of OLAP workers, causing disruption on the throughput of the OLTP execution. We tuned the proportional-integral feedback controller by experimentation to use the gain adjustment characterized by $K_P = 0.4$ and $k_I = 0.03$. The individual study on how to reach these parameters is left out of the scope of this paper. Algorithm 1 presents the Client Balancer decision process.

Algorithm 1 Client Balancer

```
 1: procedure
 2:     $wait(\Delta t)$
 3:     $error \leftarrow target\_tps - measured\_tps$
 4:     $integral \leftarrow integral + error \times \frac{1}{\Delta t}$
 5:     $output \leftarrow K_P \times error + K_I \times integral$
 6:     $previous\_error \leftarrow error$
 7:
 8:     if $output > (target\_tps \times margin)$ and $\neg saturated$ then
 9:         $start\ OLAP\ worker$
10:     else
11:         $saturated \leftarrow true$
```

The Client Balancer will periodically (Δt) poll the engine regarding the current number of transactions being delivered. This information is then fed into the feedback controller. To launch another OLAP worker, the Client Balancer ensures that the output value produced by the feedback mechanism is within a given error margin of the configured target tps and that the system is not saturated. The point of saturation is reached when the current number of OLTP transactions per second being delivered drops below the chosen threshold.

Density Consultant & Loader

HTAPBench follows the standard TPC-C transaction mix. The Density Consultant computes the correct density according to the chosen target tps and transaction mix. During the populate stage, in order to generate timestamps that follow the required density, the Loader is equipped with a clock that initiates with the system time at which the populate stage is initiated. The clock then computes how much time should elapse between clock ticks (ΔT_S) in order to fulfill the required density, as defined by expression 8.

$$\Delta T_S(ms) = \frac{1}{d(T_S/s)} \times 1000 \qquad (8)$$

The HTAPBench Loader will proceed to create and load all the table entities represented in the hybrid data schema, built from merging TPC-H's schema into TPC-C's. The final installation will scale in size according to the computed number of warehouses. When loading tables with references to date items, the Load worker makes use of the clock, increasing one tick for each new date field to be loaded. After completion, applying the density function ensures that the temporal density in the date fields matches the observed density during execution of the transactional workload.

Dynamic Query-H Generator

The analytical queries within HTAPBench are constructed according to the TPC-H specification, which requires them to be built with randomized parameters within given boundaries. The Dynamic Query-H Generator module is responsible for building the SQL statements for the queries, ensuring that the random values comply with the TPC-H specification. This module integrates with the Client Balancer module that will launch the analytical workers, with its output afterwards fed into the TPC-H worker. The Dynamic Query-H Generator computes the time window frames that should be considered for query execution, and introduces a feature that enables the window frame generation to reflect a sliding window behavior. In the specification for each query, TPC-H requires static time frames. Take as an example query `Q6` that computes the total revenue for orders placed within a given period.

Listing 1: TPC-H Query 6

```
select sum(ol_amount) as revenue from
    order_line where ol_delivery_d between
[Date] and  [Date + 1 year] and
ol_quantity between [Amount a] and [Amount b]
```

This particular query restricts the result set to orders placed within a one-year time frame, starting on January the first of a randomly selected year between 1993 and 1997, and ending in the following year. In TPC-H, this is attainable since it does not consider database growth. However, in HTAPBench, the database grows at the pace dictated by the OLTP execution of the benchmark. Thus, if window frames were to be kept static, the new regions on the dataset would never be queried. To produce a homogeneous result set that is representative of the whole dataset, the Dynamic Query-H Generator ensures that queries comply with the time range imposed by the specification while simultaneously leveraging the Density Consultant module to shift the starting date of the range to meet the speed at which the OLTP execution is making the database grow. Hence, the sliding window behavior not only ensures that the entire dataset is considered but also that consecutive executions of the same query are kept comparable.

Results Monitor

The Results Monitor collects the execution results produced by each worker. The final measurements are only collected after the configured execution time elapsed and all the workers finalized all procedures. The transactional activity is characterized according to TPC-C's metric (*tpmC*) while, the analytical activity is characterized by TPC-H's metric (*QphH*). Together with both metrics, this module also outputs data from the Client Balancer, characterizing each run in terms of how many OLAP workers were launched and when as well as the result set volume and latency for each analytical query executed.

Implementation

HTAPBench is available as an open source project [1]. It was implemented in Java for improved portability and includes all the aforementioned components. The current prototype was implemented as an extension of OLTPBench [13], a framework that enables the execution of several benchmarking approaches. Moreover, OLTPBench's implementation of TPC-C does not consider the think time used during transaction execution. The TPC-C specification requires a time in which each transaction simulates (by entering a sleep stage) the time required by the terminal user to insert data, as well as the terminal's processing time (TPC-C's simulation of a real-world scenario). Consequently, as part of our extension to OLTPBench to implement HTAPBench, we introduced the think time processing stage for each transaction of TPC-C.

HTAPBench relies on JDBC to establish a connection with the engine. Since JDBC defines a standard interface to connect with several engines, it is possible to use HTAPBench with a wide range of engines, provided that they support JDBC.

2.4 Benchmark Configuration

HTAPBench has several system requirements. The machine running HTAPBench should be provisioned with a Java distribution and the appropriate database engine driver.

HTAPBench requires the user to provide the engine JDBC connection URL, the fully qualified name of engine driver class and the target number of transactions per second (*tps*). The required target *tps* for a given configuration derives from an expectation regarding the performance compliance for the SUT. The user should start with a small target *tps* value and progressively increase the target *tps* until the SUT is saturated, as it happens with the warehouse scalability in TPC-C. Other configurations are allowed, enabling customization of the workload, namely: the Client Balancer Δt and `error margin`, the `TPC-C transaction mix`, the `TPC-H query mix` and the `Execution time`. Hereafter, we briefly describe the impact of the mandatory parameters on the system.

- `Client Balancer` Δt: This parameter configures the period in seconds used by the Client Balancer module to assess if further OLAP workers should be launched. Assigning the appropriate evaluation period has a direct impact on the convergence time and precision of the benchmark. Choosing a small value may cause the system to converge to quickly, but overestimate the number of admissible OLAP clients. On the other hand, choosing a large value improves the client balancer decision by exposing it to a larger number of samples, but delays the overall process. This parameter defaults to 60 seconds, configuring a trade-off between scalability and assuredness.
- `Client Balancer error margin`: This parameter configures how sensitive to change the Client Balancer should be, setting up the range of allowed *tps* computed in percentage of the target *tps*. It has a default value of 20%, which by experimentation we consider to be a reasonable trade-off of loss OLTP throughput in favor of having OLAP capability.
- `Execution time`: This parameter configures the duration of time that each run of the execution stage of the benchmark should last. This parameter defaults to 60 minutes, configuring a reasonable execution time after the warmup stage for the Client Balancer to exercise the SUT.

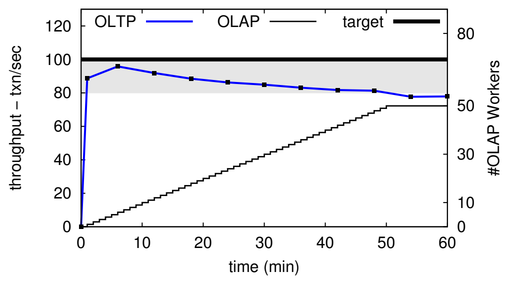
The execution of HTAPBench is divided into three main stages, as previously introduced. After system configuration, the user should run the populate script that generates the appropriate Comma-Separated-Values (CSV) files for the configured *tps*. Afterwards, the user should consult the documentation of the engine to be tested in order to use the correct procedure to load the CSV files (optionally, it is possible to directly populate the SUT, but this operation usually takes longer, since it does not load data in batch mode). To start the execution stage, which includes the initial warmup, the user should use the execution script, automatically deploying the required number of clients. The execution will run for the configured time and will afterwards produce result files, characterizing the OLTP and OLAP executions and computing the unified metric.

3. BENCHMARKING CAMPAIGN

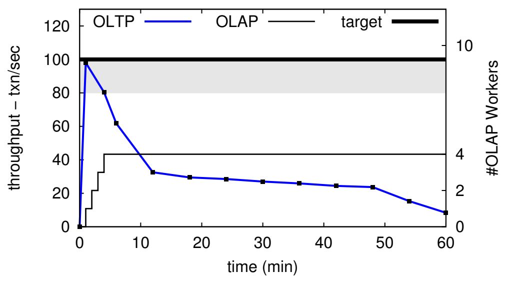
The benchmarking examples presented in this Section result from an extensive study intended to evaluate the key properties of HTAPBench. As such, the main purpose of the presented scenarios is not to quantitatively compare different SUTs, but rather to demonstrate the expressiveness of the benchmark suite and its metrics.

Three different SUT were selected, namely: (*i*) an OLTP system, (*ii*) an OLAP system and (*iii*) a Hybrid system. Besides the target *tps* and the Client Balancer error margin, the same HTAPBench configuration was used across experiments. The Client Balancer was set to use an evaluation period (Δt) of 60 seconds and the OLTP activity was regulated by the standard transaction mix within TPC-C. For the OLAP activity, we set up HTAPBench so that each business query would be chosen according to a uniform distribution. Across experiments, we configured HTAPBench to inject 100 transactions per second, which according to our density extraction mechanism amounts to 2,099 active OLTP clients and 210 warehouses, a total of 117GB of data. The selected target *tps* was chosen as the number of configured warehouses generate a dataset with over 100GB, which is above the third recommended scale factor for the OLAP agent (e.g., 1GB, 10GB, 100GB). All the following experiments reflect the average of 5 independent 60 minute runs.

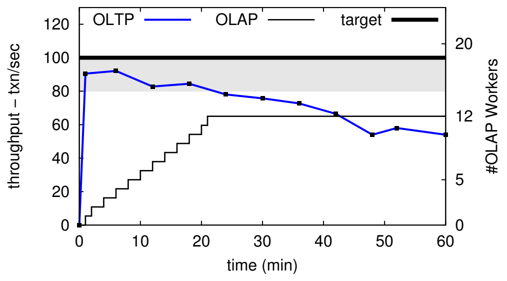
[1]https://github.com/faclc4/HTAPBench.git



(a) OLTP SUT.



(b) OLAP SUT.



(c) Hybrid SUT.

	# OLAP workers	QpH	QpHpW
OLTP	50	7	0.14 @ 756
OLAP	4	123	30.75 @ 217
Hybrid	12	169	14.14 @ 530

(d) Overview of results for different SUT.

Figure 4: Registered progress between OLTP throughput and induced OLAP scalability. The optimal trade-off between OLTP and OLAP performance is achieved when the OLTP throughput goes below the gray area, at which point the OLAP activity is not further increased. The table in (d) summarizes the OLAP results, showing that the Hybrid SUT is more balanced for an HTAP workload.

3.1 OLTP System

The current experiment used a server with an Intel Xeon E5-2670 v3 CPU with 24 physical cores (48 virtual) at 2.3 *GHz* and 72GB of memory, running Ubuntu 12.04 LTS as the operating system. We deployed an OLTP row-oriented engine and configured the level of memory that would allow the required number of clients. The Client Balancer within HTAPBench was configured to consider the default error margin of 20%. The admissible loss of OLTP throughput induced by the configuration of the error margin is depicted as a gray area in Figures 4(a), 4(b) and 4(c). HTAPBench was launched from the same machine.

The results in Figure 4(a) depict that HTAPBench was able to launch and sustain the required target *tps* throughout the entire execution time. This is read by looking at the OLTP line that is the linear interpolation of the plotted points. The required target *tps* was reached on the first minute of execution and, from that moment on, the Client Balancer started its configuration by launching an OLAP worker at each minute. The evolution regarding when and how many OLAP workers exist is read by looking at the plotted line resembling a staircase.

With a configured error margin of 20%, the bottom *tps* barrier was only surpassed after 50 minutes, saturating the system at that point in time, and therefore not launching further OLAP workers. Please consider that by default HTAPBench's Client Balancer introduces a single OLAP worker in each evaluation period (Δt). Even though we do not present this evaluation for lack of space, it is possible to configure HTAPBench's Client Balancer to deploy more than one OLAP worker at a time.

Throughout the test, a declining trend in the OLTP throughput becomes evident. In what strictly concerns the OLAP activity, the engine under test was able to hold up to 50 OLAP clients. As previously stated, the main premise in HTAPBench's evaluation scheme is to discover how many OLAP workers are required to stress out a given SUT, while ensuring that the transactional activity does not degrade beyond a configurable threshold. In addition to providing a temporal evaluation of both OLTP and OLAP workers within the system, HTAPBench outputs the unified metric we propose, but also the standard TPC-C *tpmC* and TPC-H *QphH* metrics. Within this setup, this SUT was able to sustain 756 *tpmC* and 7 *QphH*, resulting in 0.14 QpH in each OLAP worker (7 QpH / 50 OLAP workers). As such, the unified metric, *QpHpW*, amounts to 0.14 @ 756 *tpmC*.

3.2 OLAP System

The current experiment reused the previous configuration. We deployed an OLAP column-oriented engine, only setting up the required level of clients allowed in the engine.

The results in Figure 4(b) depict that the SUT sustained the OLTP throughput for a shorter period. This behavior is not unreasonable since the focus of this engine is not on OLTP activity. From the moment the threshold was broken (6th minute), the Client Balancer stopped releasing new OLAP clients. From this time on, albeit at a lower throughput, the OLTP activity was kept stable until the end of the run time. The SUT was able to register 217 *tpmC* while sustaining 4 individual OLAP clients. Cumulatively, the OLAP activity reached a peak of 123 *QphH*. As such, the unified metric we propose, *QpHpW*, reaches 30.75@217 *tpmC*.

Compared with the previous system, we observe that the latter system can enable a larger number of analytic queries, despite the fact that this particular system was only able to launch 4 OLAP streams, compared to 50 OLAP streams with the first SUT. This may seem counter-intuitive, considering that we are working atop a column-based engine specifically designed for an analytical workload. However, the 4 OLAP streams in the current SUT are much more efficient when compared with the 50 OLAP streams in the OLTP SUT.

3.3 Hybrid System

Next, we use HTAPBench to characterize a Hybrid engine. As such systems are designed to scale out, they are usually made available over a distributed architecture.

We deployed the Hybrid system over 10 nodes, 9 of which are responsible for handling and storing data, and the remaining node provides coordination and other global services. Each node has an Intel i3-2100-3.1GHz 64 bit processor with 2 physical cores (4 virtual), 8GB of RAM memory and 1 SATA II (3.0Gbit/s) hard drive, running Ubuntu 12.04 LTS as the operating system and interconnected by a switched Gigabit Ethernet network.

The results depicted in Figure 4(c) reveal that the OLTP throughput reached the assigned target in the first minute of execution. From the first minute onward, the Client Balancer started to deploy OLAP streams until the OLTP throughput degraded beyond the considered error margin, which happened after 20 minutes, registering a grand total of 12 OLAP streams. From that moment on, the OLTP throughput slowly degraded over the remainder of the test duration. Cumulatively, the SUT was able to sustain 530 *tpmC* and 169 QphH. Therefore, our unified metric *QpHpW*, presents as 14.14@530 *tpmC*.

3.4 Discussion

The results of these three tests are summarized in the table in Figure 4(d). Although the OLTP SUT achieved the highest number of OLAP workers, the results in Figure 4(d) also show that it achieved the lowest OLAP performance (0.14). The OLAP workers in the OLTP engine spend most of their time waiting for the OLAP queries to be processed and therefore complete relatively few OLAP queries. On the other hand, the OLAP engine processes significantly more analytical queries, but fails to cope with the required OLTP throughput that is used by the Client Balancer to launch additional OLAP workers. Overall the OLTP engine completed 7 QpH and the OLAP engine completed 123 QpH. This results show that the OLAP SUT was better at handling the OLAP workload, but its inability to cope with the OLTP workload in the hybrid configuration harmed the overall scalability. The favored hardware configuration for the Hybrid SUT is significantly different from the OLTP system or the OLAP system, which undermines a direct comparison among them. Nevertheless, HTAPBench was able to evaluate the Hybrid SUT, enabling 12 OLAP streams and achieving a total of 169 OLAP queries. The results reveal that the Hybrid SUT can sustain a considerable OLTP throughput with a moderate OLAP scalability.

4. VALIDATION

In order to validate the expressiveness of our metrics, we studied the system's representativeness, workload accuracy, homogeneity, error margin variability and cost. Moreover, we discuss the benchmark portability, reproducibility and repeatability. For that matter, we used all the previous SUT configurations.

First, we show that the proposed metric can either identify systems with similar or very different goals. Second, we analyze HTAPBench's accuracy by verifying the storage traces produced in three distinct scenarios. Third, we verify that the produced query result sets are homogeneous as specified. Fourth, we conduct a variability analysis which presents the consequences of varying HTAPBench's error rate. Finally we discuss the cost of using the suite.

4.1 HTAP Unified Metric

The unified metric we propose allows us to establish a comparison across two dimensions. Figure 5 uses a quadrant field plot to visually compare the relationship between our proposed metric and the target OLTP throughput.

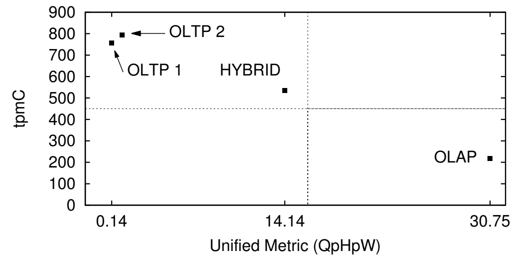


Figure 5: Quadrant field plot for the unified metric.

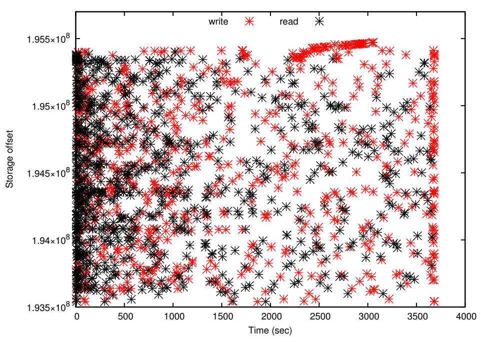
While an increase in the vertical axis translates into a higher OLTP throughput, the horizontal axis evaluates the capability to perform more analytical queries per OLAP worker. The best result would be a reading on the upper right quadrant. The analysis depicts that while the SUT labeled OLTP 1 registered a *QpHpW* of 0.14@756 *tpmC*, the OLAP observed 30.75@217 *tmpC* and the Hybrid 14.14@534.1 *tpmC*. Overall, we can state that the analyzed systems follow the trend in which the increase of OLTP activity diminishes the OLAP capability. The hybrid system reached a position close to the middle of the plot, indicating better support for the mixed workload with simultaneous OLTP and OLAP activity.

So far, the presented results allow us to conclude that the benchmarking suite is able to compare systems with very distinct work plans. In order to verify if the suite is able to distinguish systems designed for similar workloads, we reproduced the same experimental scenario as in Section 3.1 but changed the SUT to a different engine designed for OLTP workloads. We evaluated its performance and plotted it in Figure 5 with the label "OLTP 2". The results place OLTP system 2 very close to system OLTP 1 with enough precision so as not to overlap both results.

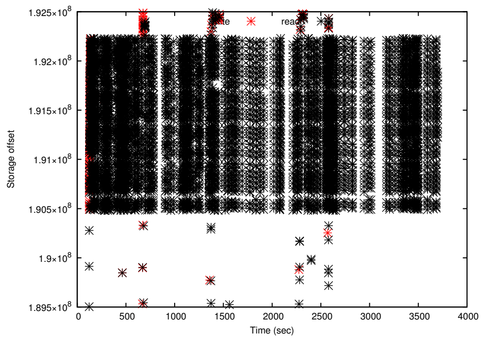
All of the previous experiments were integrated in a statistical study from where we were able to compute the variation coefficient of the computed metrics, pointing to a variation of 1.2%. As such, if the evaluation of 2 different systems, or system configurations produce variabilities with less than 1.2%, the systems should be considered indistinguishable.

4.2 Error Margin Variability

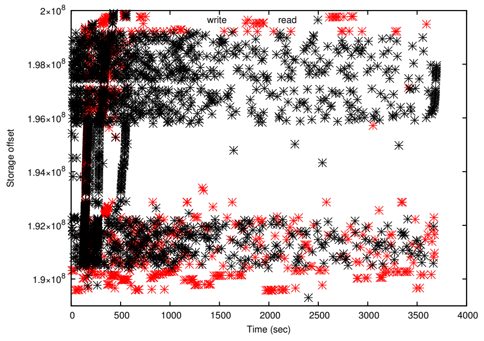
The Client Balancer in HTAPBench controls whether fur-



(a) OLTP access pattern.



(b) OLAP access pattern.



(c) HTAP access pattern.

Figure 6: Traces collected through the blktrace tool during a HTAPBench execution over an (a) OLTP, (b) OLAP and (c) HTAP SUT. The trace represents read (black marks) and write (red marks) patterns induced on the storage medium by the workload. The vertical axis represents the storage medium offset. The horizontal axis represents time in seconds.

ther OLAP streams are deployed or not, evaluating at each point in time if the current OLTP throughput does not go below a threshold defined by the error margin configuration. Intuition leads us to believe that by increasing the error margin, the Client Balancer will naturally allow more OLAP clients to be deployed, thus increasing the overall amount of analytical queries performed.

Error Rate	*QpH*	*tpmC*	*OLAP Clients*
10 %	70.99	130.64	1
20 %	168.9	131.36	5
40 %	265.99	138.30	11

Table 3: Client Balancer error rate variance.

The current experiment assesses if in fact such behavior translated into the actual behavior of the benchmark. For this purpose, we set up the OLAP SUT as in Section 3.2 but varied the error margin across runs.

First, by analyzing Table 3, the reader should be aware that the consecutive executions registered similar OLTP throughputs as expected. Moreover, as we increase the allowed error rate, we verify that more analytical queries were performed, which is a consequence of having more OLAP clients on the system, thus increasing the registered QpH and consequently the QpHpW.

4.3 Workload Representativeness

Database systems must be evaluated with representative workloads that test realistically the storage back-end capabilities. To better understand the representativeness of HTAPBench we analyzed the storage traces produced. The 3 previously tested SUT were evaluated with these workloads and the resulted storage traces were collected and analyzed with the *blktrace*[2] tool. This tool allows us to collect storage traces for a given block device. With these traces it is possible to extract useful information, such as the access patterns of storage requests, the ratio of storage reads and writes, the throughput and latency of each request, etc. Each SUT was deployed in a single machine as in Section 3.1.

Figure 6 depicts the access patterns registered by the storage medium for an HTAP SUT. The figures plot the offset of the storage medium (vertical axis) being accessed during execution time of the benchmark (horizontal axis). The analysis of the traces collected allowed us to confirm that OLTP workloads are dominated by random storage accesses (Figure 6(a)) while OLAP workloads do mostly sequential storage accesses (Figure 6(b)) dominated by read-only requests. On the other hand, mixed workloads generated by HTAPBench present a mix of these access patterns, as expected (Figure 6(c)). Moreover, the ratio of storage reads and writes is according to the specification of each workload. In summary, these results show that HTAPBench is able to simulate a realistic storage load for a hybrid SUT that pro-

[2]blktrace manual: http://linux.die.net/man/8/blktrace.

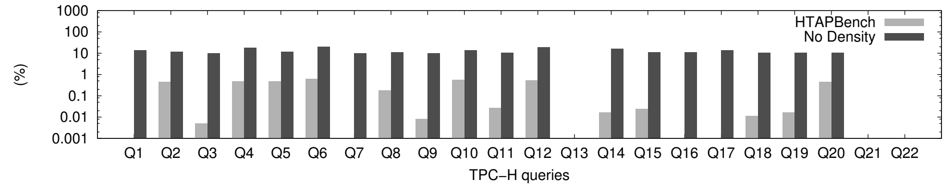


Figure 7: Result set execution cost as a percentage of the baseline across independent runs. The vertical axis is presented in logarithmic scale. The absence of columns represents null variability.

cess both OLAP and OLTP workloads simultaneously. Consequently, HTAPBench's representativeness of the analytical capability on top of a sustained transactional operation is ensured by the experimental data, but also by the individual representativeness of the TPC workloads used. Moreover, the results show that current proposals for hybrid SUT must be aware that processing simultaneously OLAP and OLTP workloads generates a mix of random and sequential storage accesses. This challenge can drive novel research proposals for both hybrid engines and the back-end storage systems supporting them.

4.4 Homogeneity and Reproducibility

The present experiment validates the effectiveness of our density mechanism. Figure 7 depicts 2 different scenarios, where we compare a distribution using random field generation with the density approach we propose. The depicted results relate to a baseline comparison where all queries were executed without any of their filtering operators, thus allowing us to extrapolate the total universe of rows considered in each analytical query. The value presented for each query is the absolute difference in produced result set rows in two consecutive executions: first, with random parameters that do not follow any distribution and, second, with the density mechanism proposed in HTAPBench.

In the first scenario, the analytical queries ran with the introduction of randomized parameters; the results show that consecutive executions of such queries produced variable result sets. The measured variability of result set lines across all query executions for this setup amounted to 12%, with a registered maximum of 20%. For the second scenario, we used HTAPBench's density mechanism to generate the random parameters to be used in all analytical queries. Queries Q13, Q21 and Q22 produce variabilities of 0% as the consecutive runs always produced the same number of result set rows.

The results show that by using our approach, we were able to produce query result sets with comparable costs. Consecutive executions of the same query registered a variability of just 0.17% on average across the different SUT and different configuration settings. These results confirm a very high level of reproducibility of results for a given configuration.

4.5 Benchmark Adoption Effort

The benchmarking suite proposed in this paper derives from two industry standard benchmarks with added functionality. The available suites offer implementations that are individually deployed to collect separate metrics, and they typically require the implementation of connector mechanisms between the benchmarking suite and the engine driver.

HTAPBench follows the trend of a few other suites, relying in the native JDBC driver of the SUT to communicate all the operations of the workload. Concerning the required time to run the benchmark, the setups considered in this paper are the result of 5 independent runs. In each one of them, the SUT had to be populated with data and then tested with the hybrid workload. The time spent during the populate stage greatly depends on the configured target of transactions per second and the SUT used. In the setups considered, we observed an average of 4 hours to populate the files for the engine. Afterwards, before subsequent executions, the engine would dump and restore the initial populate data, taking an average of 15 minutes. This technique is also valid for setups with terabytes of data. Regarding the execution stage, each setup ran for 60 minutes. Consequently, the overall cost for each SUT was 10 hours. Considering other suites, the total cost per SUT is relatively the same.

5. RELATED WORK

There are several organizations that propose benchmarking standards to assess either transactional or analytical systems, namely: the Transaction Performance Council (TPC) [12], the Standard Performance Evaluation Council (SPEC) [7] and the Storage Performance Council (SPC) [6]. We categorize several systems into OLTP, OLAP or Hybrid and we briefly present their main characteristics and shortcomings.

5.1 OLTP

Online Transactional Benchmarking systems, such as TPC-C [8], focus on assessing a system's ability to cope with a large amount of small-sized transactions. Usually, OLTP systems rely on row-oriented data stores, in which the transactions operate over a restricted space of tuples, ensuring at the same time properties such as consistency and isolation in what is commonly referred to as ACID [17].

The TPC-C specification models a real-world scenario where a company, comprised of several warehouses and districts, processes orders placed by clients. The workload is defined over 9 tables operated by a transaction mix comprised of five different transactions, namely: New Order, Payment, Order-Status, Delivery and Stock-Level. Each transaction is composed of several read and update operations, where 92% are update operations, which characterizes this as a write heavy workload. The benchmark is divided into a load and an execution stage. During the first stage, the database tables are populated and, during the second stage, the transaction mix is executed over that dataset. TPC-C defines how these tables are populated and also defines their size and scaling requirements, which is indexed to the number of configured warehouses in the system. The outcome of this benchmark is a metric defined as the maximum qualified throughput of the system, *tpmC*, or the number of New Order transactions per minute.

The TPC-E [11] benchmark builds on TPC-C, introducing a variable number of client terminals. It models a scenario where a brokerage firm receives stock purchase requests from customers, trying to acquire the correspondent stock bonds from a Stock pool. The purchase orders placed by clients are based on asynchronous transactions while stock requests between the Brokerage firm and the Stock Exchange are based on synchronous transactions. Compared with TPC-C, this benchmark builds a much wider and more complex system as it is composed of 33 tables and 10 transactions, 6 of which are read-only and the remainder are read-write transactions; the latter accounting for 23% of all requests.

Both specifications build benchmarking suites strictly intended to evaluate OLTP engines. Despite their representativity of OLTP workloads, the characteristic short operational transactions prevent the workload from exercising OLAP requirements. Engines designed to provide a high OLTP throughput are typically row oriented as the nature of a single transaction induces I/O bound operations in several attributes per transaction. This behavior translates into random access patterns to the storage medium and does not produce CPU and memory bound operations as do OLAP workloads.

5.2 OLAP

Online Analytical Systems such as TPC-H [9, 25] or TPC-DS [10] focus on assessing a system's ability to perform multi-dimensional operations, usually on top of column-oriented data stores.

TPC-H builds a workload that does not require ETL, modeling a real world-scenario where a wholesale supplier must perform deliveries worldwide. The business queries perform complex data operations (e.g., aggregations) across large sets of data. The workload defines a query mix comprised of 22 business queries that access 8 different tables. The execution is divided into three stages. The first stage loads the database. During the second stage, a single user executes all 22 business queries, while during the third stage, a multi-user setup is used in order to evaluate the system's ability to schedule concurrent operations. TPC-H does not consider any growth factor during runtime, which means that the dataset does not change in terms of its total size. The outcome of this benchmark is computed through a metric that accounts for the total number of completed queries per hour (*QphH*).

TPC-DS builds a workload that requires ETL, namely to ensure data freshness. It models a scenario where orders must be processed from physical and online stores of a wholesale supplier, mapping it into a star schema composed of 7 fact tables and 18 dimensions. The workload holds 4 different query types, namely Reporting, Iterative, Data Mining and Ad-hoc queries. The database populated for TPC-DS, as in TPC-H, does not consider any growth factor; still, the initial population is regulated in terms of a scale factor that has direct influence over the data size. The output metric is defined as the number of queries per hour at a given scale factor, *QphDS@ScaleFactor*.

TPC-DS is seen as an evolution of TPC-H, addressing oversimplifications that prevent the proper evaluation of OLAP systems. However, the need to use ETL to promote updates on the star schema prevents us from using it as an OLAP agent in our hybrid workload.

The high prevalence of read operations in these workloads mostly generate sequential accesses to storage mediums, and therefore are not able to simulate the short and cross attribute nature of OLTP workloads that also live in a hybrid workload.

5.3 Hybrid

Hybrid workloads should capture both access patterns observed in the previous individual specifications [15]. There are a few benchmarking suites that use both access patterns, namely CH-benCHmark [5] and CBTR [4, 2, 1, 3].

CH-benCHmark creates a mixed workload also based on TPC standard benchmarks, enabling two types of clients. A transactional client provides a TPC-C agent, while an analytical client provides a TPC-H agent. To allow the analytical workload across the transactional relations, CH-benCHmark merged both schema into a single one, comprising relations from TPC-C and TPC-H. The relations accessed by the OLTP execution scale according to TPC-C's specification. Relations accessed by the OLAP execution are kept unchanged. However, CH-benCHmark neglects aspects within TPC-H's specification. Namely, the analytical queries should hold random parameters in order not to constantly transverse the same regions of the dataset. It also disregards the required distributions for date fields, impacting the produced analytical results.

CBTR defines a benchmarking system aimed at mixed workloads, which does not account for any previous standardized specifications, considering them too predictable. It introduces a workload built from real enterprise data that models an order-to-cash workflow. For that matter, a new schema and the respective transactional activity is presented. By using real data, CBTR bypasses the need to use numerical distributions to populate or to generate data during benchmark execution.

The major differentiator of HTAPBench when compared with the previous approaches lies specifically in its Client Balancer module that governs how both workloads coexist. Both CH-Benchmark and CBTR use naive approaches to find the maximum qualified throughput for both the transactional and analytical workloads. The main concern that arises is that neither workload agent in each benchmarking suite is aware of the other agent, creating a dispute for resources as each agent tries to saturate the SUT. The Client Balancer in HTAPBench follows Gartner's recommendation to specify how the transactional and analytical agents coexist; instructing the transactional agent to sustain a configured throughput and allowing the analytical agent to saturate the SUT up to the point when the transactional throughput is affected. Moreover, HTAPBench also addresses the issues found in CH-Benchmark relating to non-uniform result sets by using the density mechanism to regulate that behavior.

6. CONCLUSION

This paper proposed HTAPBench, a benchmarking suite for engines that support hybrid workloads composed of high levels of transactional activity and, at the same time, provide business analytics directly over production data.

As our main contributions, we proposed a unified metric that enables the quantitative comparison of very similar systems, as well as very different systems. We also propose solutions to the key challenges of a hybrid benchmark, such as the Client Balancer.

Moreover we provide homogeneous and comparable results across executions. We validated our prototype on top of OLTP, OLAP and Hybrid systems, demonstrating the inherent expressiveness of HTAPBench's metrics to characterize such systems. We show that HTAPBench is able to distinguish different classes of systems (i.e., OLTP from OLAP), as well as distinguish systems within the same class with high precision. We ensured that HTAPBench exercised the storage layout as expected for each workload type. The results allowed us to conclude that by using the proposed approach we were able to introduce the required workload randomness while keeping the results comparable by ensuring equal query execution costs across the whole dataset.

7. ACKNOWLEDGMENTS

The authors would like to thank Marco Vieira for all the constructive reviews and comments on the final stage of this work, but also to Martin Arlitt and the anonymous reviewers for their helpful comments. This work is financed by: (1) the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalization - COMPETE 2020 Programme within project POCI-01-0145-FEDER-006961, and by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia as part of project UID/EEA/50014/2013; (2) the European Union's Horizon 2020 - The EU Framework Programme for Research and Innovation 2014-2020, under grant agreement No. 653884.

8. REFERENCES

[1] A. Bog, J. Krüger, and J. Schaffner. A composite benchmark for online transaction processing and operational reporting. In *Advanced Management of Information for Globalized Enterprises, 2008. AMIGE 2008. IEEE Symposium on*, pages 1–5. IEEE, 2008.

[2] A. Bog, H. Plattner, and A. Zeier. A mixed transaction processing and operational reporting benchmark. *Information Systems Frontiers*, 13(3):321–335, July 2011.

[3] A. Bog, K. Sachs, and H. Plattner. Interactive performance monitoring of a composite oltp and olap workload. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 645–648, New York, NY, USA, 2012. ACM.

[4] A. Bog, K. Sachs, and A. Zeier. Benchmarking database design for mixed oltp and olap workloads. In *Proceedings of the 2Nd ACM/SPEC International Conference on Performance Engineering*, ICPE '11, pages 417–418, New York, NY, USA, 2011. ACM.

[5] 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.

[6] S. P. Council. *Storage Performance Council*. 2015.

[7] S. P. E. Council. *Standard Performance Evaluation Council*. 2015.

[8] T. P. P. Council. *TPC Benchmark C*. 2010.

[9] T. P. P. Council. *TPC Benchmark H*. 2010.

[10] T. P. P. Council. *TPC Benchmark DS*. 2012.

[11] T. P. P. Council. *TPC Benchmark E*. 2015.

[12] T. P. P. Council. *The Transaction Processing Performance Council*. 2015.

[13] 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.

[14] C. D. French. "one size fits all" database architectures do not work for dss. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, SIGMOD '95, pages 449–450, New York, NY, USA, 1995. ACM.

[15] F. Funke, A. Kemper, and T. Neumann. Benchmarking hybrid oltp&olap database systems. In *BTW*, pages 390–409, 2011.

[16] A. K. Goel, J. Pound, N. Auch, P. Bumbulis, S. MacLean, F. Färber, F. Gropengiesser, C. Mathis, T. Bodner, and W. Lehner. Towards scalable real-time analytics: An architecture for scale-out of olxp workloads. *Proc. VLDB Endow.*, 8(12):1716–1727, Aug. 2015.

[17] T. Haerder and A. Reuter. Principles of transaction-oriented database recovery. *ACM Comput. Surv.*, 15(4):287–317, Dec. 1983.

[18] R. Kimball, M. Ross, et al. The data warehouse toolkit: the complete guide to dimensional modelling. *US: John Wiley & Sons*, 2002.

[19] S. S. Lightstone, T. J. Teorey, and T. Nadeau. *Physical Database Design: The Database Professional's Guide to Exploiting Indexes, Views, Storage, and More*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[20] S. Machine. Splice machine. Technical report, Splice Macine, 2015.

[21] N. Mukherjee, S. Chavan, M. Colgan, D. Das, M. Gleeson, S. Hase, A. Holloway, H. Jin, J. Kamp, K. Kulkarni, T. Lahiri, J. Loaiza, N. Macnaughton, V. Marwah, A. Mullick, A. Witkowski, J. Yan, and M. Zait. Distributed architecture of oracle database in-memory. *Proc. VLDB Endow.*, 8(12):1630–1641, Aug. 2015.

[22] M. Nakamura, T. Tabaru, Y. Ujibashi, T. Hashida, M. Kawaba, and L. Harada. Extending postgresql to handle olxp workloads. In *Innovative Computing Technology (INTECH), 2015 Fifth International Conference on*, pages 40–44. IEEE, 2015.

[23] R. O. Nambiar and M. Poess. The making of tpc-ds. In *Proceedings of the 32Nd International Conference on Very Large Data Bases*, VLDB '06, pages 1049–1058. VLDB Endowment, 2006.

[24] NuoDB. Hybrid transaction and analytical processing with nuodb. Technical report, NuoDB, 2014.

[25] M. Poess and C. Floyd. New tpc benchmarks for decision support and web commerce. *SIGMOD Rec.*, 29(4):64–71, Dec. 2000.

[26] J. Rossberg and R. Redler. *Pro Scalable .NET 2.0 Application Designs*. Expert's Voice in .Net. Apress, 2005.

[27] VoltDB. Voltdb - whitepaper. Technical report, VoltDB, 2014.

[28] M. Waclawiczek. *HTAP - What it Is and Why it Matters*. 2015.