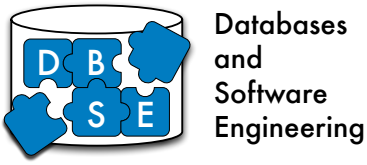
Otto-von-Guericke-Universität Magdeburg

Faculty of Computer Science



Master's Thesis

Towards Including Freshness Measures in HTAP Benchmarks

Author:

Harita Medimi

May 23, 2018

Advisors:

MSc. Gabriel Campero Durand

Data and Knowledge Engineering Group

Prof. Dr. rer. nat. habil. Gunter Saake

Data and Knowledge Engineering Group

Medimi, Harita:
Towards Including Freshness Measures in HTAP Benchmarks
Master's Thesis, Otto-von-Guericke-Universität Magdeburg, 2018.

Abstract

Database workloads are traditionally divided into two families, *analytical* and *transactional*, each with their own set of matching requirements and optimizations. In recent years, *hybrid transactional and analytical* (HTAP) systems have been developed to support these two kinds of workloads within a single system, through careful light-weight replication and synchronization. The design of benchmarks for these systems poses several challenges: a) the need for measuring the goodness of the system under test to *isolate* OLTP workloads from the intense resource consumption of concurrent OLAP clients on the same machine, b) the requirement of having a *unified measure that could characterize performance across different combinations of workloads*, c) the need for reporting an informative measure of the actual *freshness* of the OLAP reads.

In this Master Thesis we review the state of the art in HTAP benchmarks, complementing this with a practical evaluation using the HTAPBench benchmark, a tool that seeks to address the first 2 challenges mentioned.

As our contribution to this field we focus on the third challenge: *reporting an informative measure of the actual freshness of the OLAP reads*. For this, we study and evaluate on *freshness measures* proposed in the literature; using a simple prototype for an HTAP database that relies on replication from OLTP to OLAP formats, and that supports reads with bounded staleness. We run the YCSB benchmark, an established OLTP benchmark that enables to set a limited amount of concurrent updates and read operations, over our prototype, and report the throughput and freshness measures at various refresh rates for the replicated data. With our experiments we can identify Absolute Freshness and Freshness Rate as the preferred choices for measures that can use information posterior to the replica update time. This is the case for benchmarks. We also identify Absolute Timeliness as the best metric for cases that cannot employ information posterior to the update to the replica, which could be the case for query engines of HTAP DBMSs. In addition, we propose and test a novel metric, Timeliness Rate, to capitalize on volatility models for approximating the Freshness Rate, under limited information of ongoing updates.

We expect our work to contribute to the development of standards for measuring freshness by providing practical evaluations of this data quality dimension. We hope that our observations could towards building query engines able to reason about freshness in any situation where replicated data management is needed, like HTAP systems or co-processor accelerated systems.

Acknowledgements

By submitting this thesis, my long term association with Otto Von Guericke University will come to an end.

First and foremost, I would like to thank Prof. Dr. rer. nat. habil. Gunter Saake for giving me the opportunity to write my Master's thesis at his chair.

I am grateful to my advisor MSc. Gabriel Campero Durand for his time to time guidance, infinite patience and constant encouragement without which this may not have been possible. It has been a privilege for me to work in Data and Knowledge Engineering Group.

I would like to thank Pavlo Shevchenko, Ali Hashaam and Guzel Mussilova, developers (in the context of a scientific team project at the DBSE workgroup) of Blinktopus, the small HTAP prototype which I adapted for my evaluations and instrumented for the diverse freshness measures.

I would like to thank my best friends Priya Palaniswamy, Ramya Dirsumilli, my parents and my husband, who supported me in completing my studies and in writing my thesis. I can not imagine completing my Master studies without them. I am thanking them all from the bottom of my heart.

Declaration of Academic Integrity

I hereby declare that this thesis is solely my own work and I have cited all external sources used.

Magdeburg, May 23rd 2018

_____

Harita Medimi

Contents

List of Figures

1. Introduction

In the past decades, operational and analytical systems have been supported in relative isolation, by specialized DBMSs. This architecture has resulted in systems with decent performance. However, to maintain both systems separate, enterprises and organizations have to bear a cost higher than that of maintaining a single system. In this type of architecture, the data needs to be recorded in a transactional system, and it can only be analyzed after being copied to an analytical system, often through a heavy ETL process.

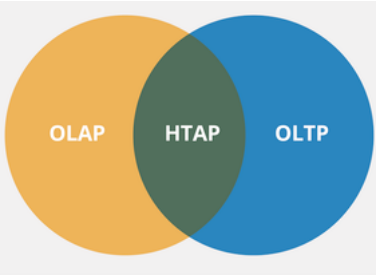


Figure 1.1: Hybrid Transactional Analytical Processing [Web18]

In todays modern world, it is highly important to do analysis on real time data as early as possible, in order to achieve advantage over competitors in the business world. Therefore, Hybrid Transactional and Analytical Processing systems (HTAP)[1] are emerging as a solution with a crucial role for enterprise data management. These systems have very diverse design characteristics that enable them to support well both online transactional processing (OLTP) and online analytical processing (OLAP). In HTAP systems there is no need of maintaining multiple copies of data, furthermore, analytics can be carried out on the most recent real-time data within a single system. This last feature is the most distinguishing one of HTAP DBMSs, it can represent a business advantage for users of these systems who are able to leverage such feature.

[1]The word HTAP is coined by Gartner [Gar14].

The traditional database workloads, OLTP and OLAP, are used for serving the transactional and decision making applications of organizations, respectively. OLTP systems are focused on providing high throughput for a large amount of small operations, while OLAP systems main goal is to power the analysis of huge amounts of data and support the intended decision making process. To evaluate the performance of these systems there are different benchmark standards which are offered by Transactional Processing Performance Council (TPC). The list of benchmarks provided by TPC include TPC-C, TPC-E, TPC-H ,TPC-DS and OLTP. These report the performance, throughput and latency for either OLTP or OLAP systems. The high demand for evaluating systems that support mixed workloads (HTAP) necessitates the evolution of mixed workload benchmarks, which are crucial to help these technologies develop by enabling the comparison of systems, helping the community to realize which design features are useful and which not.

Authors have studied that the performance of HTAP DBMSs [PWM+14] depends on specific factors such as the freshness of the OLAP reads, the flexibility of the system to rewrite queries, and the scheduling of resources among both workloads. Authors report a "house pattern" as a standard occurrence for HTAP systems, whereby an increase in OLAP clients leads to a loss of throughput for the concurrent OLTP clients[2]. As a consequence of such factors, specialized testing is required to assess the goodness of HTAP systems in isolating both workloads and providing good performance at a given expected freshness.

Specifically, the design of HTAP benchmarks present several challenging requirements, as discussed by authors in the field [Bog12] [CFG+11] [CPV+17]:

- *Providing an evaluation for workload isolation:* To measure the capabilities of the system for shielding OLTP workloads from the high resource requirements of concurrent OLAP clients running on the same machine.
- *Offering a unified performance measure:* HTAP benchmarks should be able to report a single unified measure to represent the performance of the system under certain combination of workloads and proportions of the representative queries per workload.
- *Characterizing freshness:* One of the fundamental design choices of HTAP systems can be the support of OLAP reads with less freshness. This means that the strong requirement for transactional serializability, usually supported through so-called *strong reads*, can be lowered to enable OLAP clients to read on data replicas which are slightly stale and not up to date. This is also called relaxed serializability [BFG+06] or enabling reads with *bounded staleness* (as used in big data systems [CCH+14]). Such a design can result in higher performance for the OLAP clients. Stemming from the potential gains from this design, HTAP systems have moved beyond designs that offer the same bounded staleness to all

[2]This was also reported by Bog in the context of evaluating CBTR [BPZ11].

reads [KNF+12], to offer each request the ability to specify their own staleness bound [BBB+17] [LMHK+17]. As a result, HTAP benchmarks need to be extended such that they report the freshness for OLAP queries in their evaluations. This is a significant challenge since, though a large number of measures have been proposed [OC12], there is currently no standard metric for reporting freshness.

- *Extending scale factor concepts with scale changes:* Unlike OLAP-only systems, HTAP systems can experience database growth and evolution from one scale factor to another. As a result HTAP benchmarks should be able to evaluate the goodness of the system under test to handle changes in database scale factor.

- *Additional support for ad-hoc queries:* Since OLAP queries can go beyond the traditionally used benchmarks like TPC-H, it can be useful to support ad-hoc queries in HTAP benchmarks.

Due to this set of challenges, next to the relative novelty of HTAP DBMSs, to date there is no standard HTAP benchmark with mainstream adoption, and more particularly, no standard for evaluating freshness. Such situation might hinder the development and adoption of the technologies, leading to untapped potential.

Tools like CBTR [BPZ11], CH-BenCHmark [CFG+11] and HTAPBench [CPV+17] have been proposed as mixed workload benchmarks. Each of them extends a bit the coverage in requirements for these benchmarks. CBTR considers a real-time business scenario different than the traditional TPC schemas. In their studies, authors employ the tool to evaluate the impact of different schema normalizations for mixed workloads [BPZ11]. CH-BenCHmark covers all the queries from TPC-H and TPC-C over a unified schema. The tool provides users with the facility of configuring different workload proportions and to define a single read staleness bound [CFG+11]. To an extent HTAPBench builds on the work of CH-BenCHmark, considering the same schema and workload possibilities [CPV+17]. HTAPBench adds to useful contributions to the field. First, it suggests a scheme based on a feedback controller, to balance the maximum number of OLAP clients scheduled while keeping OLTP throughput for a fixed number of clients within a threshold. Authors also propose a novel unified measure to evaluate the system under test, and an informative visualization that conveys relevant features of how OLAP and OLTP clients interact for the given system under test.

In spite of their contributions, these tools fail to address the challenge of characterizing the freshness of the system under test, providing no measure for it. Furthermore, they have no concept, as of the time of this writing, for evaluating workload that contain individual reads with varying bounded staleness.

In consideration that the data quality dimension of freshness is an important factor for HTAP systems, in this study we propose to contribute towards characterizing freshness in HTAP benchmarks. To this end we evaluate a comprehensive number of 12 freshness/staleness measures proposed in the literature [RBSS02, GLRG04, Bou04, NLF05, BFG+06, QL07, HKK09, OC12, CCH+14], by adopting a small prototype for

an HTAP system, and extending a benchmarking configuration such that the different measures can be recorded. We provide practical results for how these measures portray the system under test over one configuration of the benchmark, and we found that the metrics which are global measures and time-dependent metrics are not useful. We show that some of the proposed measures are not applicable to our use case. We recommend metrics that provide reasonable insights. We propose an new metric, Timeliness Rate, which uses the modeled volatility of items to predict the freshness of the read replica as a whole.

Moreover, we include a practical evaluation of the existing benchmarks, considering the information that they provide in their detailed reports. We discuss, as well, how the freshness measures could be adopted in them, apart from deciding on their utility and cost for recording them.

Thus, we can summarize our contributions as follows:

- We provide necessary background to understand the state of the art for HTAP benchmarks.
- We adapt an HTAP prototype based on replication across formats, such that it can be used with the YCSB benchmark, creating a log of all the requests it receives. We implement, as well, a tool that based on the generated logs, calculates diverse freshness measures proposed in the literature. Hence we give a comprehensive profile for the freshness of reads in the reported performance results the benchmark.
- Our tool can be used for any logs that share our current reporting format. Our log generator can be ported to other YCSB clients and systems.
- We compare, for a specific scenario with mixed requests, the results for the generated measures. We evaluate their ability to represent the precise freshness characteristics provided by the system under test, and we report in addition about how they compare in terms of required parameters to be kept while running the benchmark. We find which metrics are more informative than others, and we propose a new metric to extend those we study.
- We provide a practical walk-through of the work with two of the open source available tools, discussing the features in their reports and limitations. We discuss how the freshness measures that we evaluate could be added to these tools.

1.1 Structure

The structure of this Thesis is as follows:

- **Background:** In this chapter we present the necessary theoretical background and context for our study. We cover benchmarking, HTAP systems, HTAP benchmarks and freshness concepts (Chapter 2).

- **Prototypical Implementation:** We present the various methods for measuring the freshness and evaluating with the YCSB benchmark. We describe, as well, the prototype HTAP system that we used for our tests (Chapter 3).
- **The Evaluation of Freshness:** We present the results for our evaluation about freshness, how it affects the performance of the mixed workloads, and how the freshness measures represent the system under test (Chapter 4).
- **Results from HTAP Benchmarks and Alternatives for Adding Freshness:** We provide a simple and practical evaluation of the use of state of the art HTAP benchmarks. We discuss how freshness could be added to them (Chapter 5).
- **Conclusion and Future Work:** We summarize the results of our study, threats to the validity of our work, and we provide an outlook for future work (Chapter 6).

2. Background

In this chapter we present the necessary theoretical background and context for our study. We organize this presentation as follows:

- **Context for the Development of HTAP** We start by giving a brief description about HTAP, with a short introduction to OLTP (Online Transactional Processing) and OLAP (Online Analytical Processing) workloads. We also give an overview of existing HTAP systems and key characteristics.
- **Benchmark** Next we consider benchmarking for database systems, with a short history about benchmarking (Section 2.2) and a discussion on classification of benchmarks (Section 2.2.1). Following TPC practice, we further divide the benchmarks into transaction processing benchmarks(Section 2.2.2), analytical processing benchmarks (Section 2.2.3)and mixed workload benchmarks. Our discussion on HTAP-specific topics is based on the following selection of primary studies:
  - Tutorials on HTAP systems [BDM+16, OTT17].
  - Papers on benchmarking for HTAP [CFG+11, Bog12, PWM+14, CPV+17].
  - Papers that present the implementation of HTAP systems, or of components of them [GKP+10, KN11, FCP+12, KNF+12, MGBA17, BBB+17, LMHK+17].
  - Papers describing freshness metrics for different data management uses [RBSS02, GLRG04, Bou04, NLF05, BFG+06, QL07, HKK09, OC12, CCH+14].

2.1 Hybrid Transactional and Analytical Processing

In this section we introduce OLAP, OLTP and HTAP. We do not intend to provide a complete historical coverage of the topics. Instead we focus on introducing them for our ensuring discussion on HTAP benchmarking.

Relational database systems have played a significant role in business applications for many years. All businesses need data management systems to handle various parts of their daily activities which include manufacturing, sales and orders, financial management and the analysis of data for strategic decision making etc. In the past, a majority of these database systems have been designed to handle online transaction processing in which the database receives a workload made up of a large amount of transactional operations, these consist of many singular tuple-level insertions, reads or updates (e.g. corresponding to the sales of an item in e-commerce, or the updating of a profile in a social network). The majority of database providers came up with specialized designs of data structures and architectures to manage such workloads. Data management systems designed to handle these workloads are called Online Transactional Processing (OLTP) systems. H-Store is an example of a recent OLTP system, constituted by single-threaded engines (i.e., workers) which provide lock-free execution of OLTP transactions. [KKN+08].

Coupled with the use case for OLTP, for the past 20 years, the employment of specialized techniques of data analysis for decision making has gained importance, with terms like business analytics or data science used to group these techniques . Specialized systems have been developed to support most applications of the techniques, these are known as Online Analytical Processing (OLAP) systems. Such tools are used for multi-dimensional views, with historical considerations, of business activities. MonetDB is an example of an OLAP system, and it is used in the field of health care, telecommunication and astronomy [IGN+12].

OLAP and OLTP workloads are different from each other in the complexity of the queries that constitute them, in the expected processing speeds, in the most fitting database design, and in the resource requirements, among other features.

Characteristics	OLTP	OLAP
Queries	Simple queries	Complex queries
Processing speed	Very fast	Takes many hours
Database design	Normalized with many tables	De-normalized with fewer tables
Space requirement per query	Small number of tuples	Large space due to historical data

Table 2.1: OLAP vs OLTP

Typically OLAP systems and OLTP systems work separately in such a way that analytical workloads do not affect the concurrent transactional workloads. In this approach, analytical queries run on a data warehouse with replicated data from one or more operational systems. Often OLAP systems consume data that is not up-to-date.

Plattner [Pla09] stated that advancements of in-memory technologies have enabled a new approach for standard business applications where data can be stored just once without compromising either transactions or analytical workloads. This new kind of systems are now commonly know as **H**ybrid **T**ransactional and **A**nalytical **P**rocessing (HTAP) systems, as named by Gartner [Gar14]. Some characteristics of these systems are discussed next.

- No ETL: For analytics in HTAP, there is no requirement for data transfer from operational databases to data warehouses.
- Data freshness for analysis: Transactional data is readily available for analytics as soon as it is created.
- Freshness of aggregates and pre-computed values: Drill-down operations from analytic aggregates always points to fresh HTAP application data.
- Less storage requirements: It eliminates or at least reduces the need for multiple copies of the same data.

HTAP databases could play a considerable role in enabling freshness of operational data for decision making, hence they could have great beneficial impact in these different industries. Decision making on real time data could be a potentially valuable capability for different organizations. Public safety, risk management, fraud detection, and others might reasonably benefit from such capability.

2.1.1 Different HTAP systems

Tutorials presented at diverse database conferences have covered HTAP workloads and database systems designed for them. To our knowledge such tutorials are limited to the presentations by Bohm et.al[BDM+16] and by Ozcan et.al [OTT17]. The first covers the design of some HTAP systems, both academic and industrial, such as SAP HANA, Oracle InMemory, MemSQL and Cloudera Impala with Kudu. Apart from presenting the design of the systems this tutorial does not offer a listing of open challenges. There different HTAP systems vary on storage model, architecture, updates and layouts. A summary of different HTAP systems is given in the below table.

System	Architecture	Layout	Update Mechanism
Hyper (2010)	Single System	Static. Either row or column for all tables. Horizontal partitions	Snapshot mechanisum
Hyrise (2010)	Single System	Dynamic. Complete DSM with different vertical grouping	Delta buffers
OctopusDB (2011)	Single System	Log store, Static per view. Row, Column, PAX, Index	In-place, propagated from the log
SAP HANA (2012)	Single System	Dynamic: Row for Hot data Columns for Cold Horizontal partitions.	Delta buffer
ES2 (2011)	Single System on a clustered file system	Static PAX layout Horizontal partition	Delta buffer or One simple write per transaction
MemSQL (2013)	Single system	Static per table: rows or columns Columns are stored with PAX Horizontal partition	Dalta buffers
H20 (2014)	Single system	Dynamic through vertical partition	-
Peloton (2017)	Single System	Dynamic Horizontal partitions with vertical sub-partitions	In-place update in MVOCC
Batch DB (2017)	Single System	Static. Horizontal partitions row-wise	Snapshot replica, Updates are interleaved with queries at OLAP components
Google Spanner (2017)	Single System	Static : PAX, Horizontal partitions with co-location and eager replication	Message Passing
IBM Wildfire (2017)	Tight coupling of Wildfire and Spark	Static, PAX. Horizontal Partitons	Delta buffers

Table 2.2: Synopsis of Different HTAP Systems

2.1.1.1 Hyper

Hyper is a main-memory database system which can handle OLAP and OLTP workloads by using a consistent virtual memory snapshots mechanism. This mechanism happens at system-configured time intervals. [KN11].

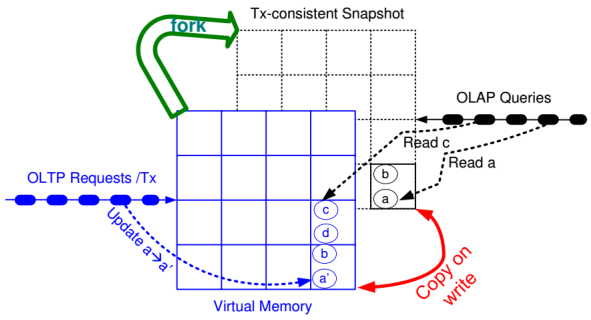


Figure 2.1: HyPer: Virtual Memory Snapshot of OLAP &OLTP[KN11]

Architecture: As we know, to consolidate the two disparate workloads OLTP and OLAP on to one system is a highly challenging task. In HyPer, hardware-assisted replication mechanisms are used to maintain consistent snapshots of transactional data [KN11]. HyPer processes the OLTP queries with lock-free control and guarantees ACID properties. To execute the analytical queries, it uses virtual memory snapshots. The OLAP snapshots are obtained by a fork system call from the OLTP process. The fork system call creates a child process which is an exact copy of the parent process. As soon as an update happens, new memory is allocated for the parent process, such that it has new values for variables, on the other hand, the OLAP process holds the old values with the previous memory locations. The OLAP queries are executed on the snapshots (i.e., the child processes) with a running parallel OLTP workload in one system. According to some studies, the throughput of the transactional workload for the HyPer system is better than for specialized OLTP engines like VoltDB. The query response time of Hyper is very fast compared to other OLAP engines[KNF+12].

2.1.1.2 Hyrise

Hyrise is a hybrid database system which is based on vertical partition of different widths [GKP+10]. Through the partitioning it seeks to reduce I/O costs for operations and improve the memory use.

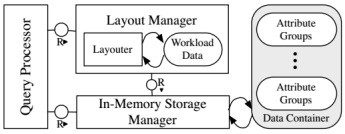


Figure 2.2: Hyrise Architecture[GKP+10]

Architecture: The Hyrise architecture is mainly divided into three parts: the query processor, the storage manager and the data container. This is similar to several database systems, but the storage manager has different features for handling data containers. The query processor prepares a physical query plan based on the user queries. The tasks of the storage manager include creating and managing the hybrid containers with data. The storage manager includes a layout manager which suggests the best possible layout design for the workload provided by the Query Processor. Hyrise proposes an algorithm for vertical partitioning, such that the best grouping of columns is formed for query processing.

2.1.1.3 SAP HANA

SAP HANA is an in-memory DBMS [FCP+12]. Through the years this system has evolved to include diverse features. As of now it is a multi-processing engine which supports the classical relational data with both row and column layout. It also provides features for efficient large scale processing with Spark (introduced with SAP HANA VORA), graph processing (SAP HANA GRAPH), search engines and others.

Architecture: The center point of the SAP HANA architecture is an in-memory processing engine. The relational data is stored in a table, which resides in main memory. The physical representation of data can be as column or row layout, and can be changed from one layout to other layout. By default, data enters the system as a row and ages towards columns. For efficiency the columnar data is dictionary encoded, with compression also applied, leading to good memory uses and enabling data skipping during query processing. As SAP HANA also supports graph and text data, they can provide storage in respective graph and text formats[IGN+12].

Octopus DB[DJ11] stores the data in a log form. This log contains both committed and uncommitted data. Then a Storage View (SV) concept is used for committed data, these are replicas of part of the data, and they are defined by queries. By using SVs, similar to the use of materialized views, less data needs to be considered to answer a query. SVs contain data sections in either row, column, PAX pages or index form and it can also describe the whole log data as SV. It also creates suitable SV according to the different workloads,i.e. defined as the *Holistic Storage View Optimizer*. According to the query and considering the cost factor, it decides to create new SV or search in the log. It follows the mechanism of Multiversion Concurrency control and it also holds the ACID property.

2.1.1.4 Others HTAP Systems

MemsSQL is one of the distributed SQL [CJW+16] databases. In this system the transactional data is stored in a row format in memory and for the analytical processing this data is converted to column form in disk. Here, for row formats skiplists are used to access data and column stores means that PAX-like segments are used. It updates as batches periodically merging the data to column store. It uses the multi version optimistic concurrency control which avoids conflicts among transactions.

Peloton is an open source database management system, that uses a flexible storage model. A logical tile abstraction lets the DBMS do query execution plans on data with minimal overhead. It was proposed in [APM16]. Every table's physical design to an evolving query workload is enhanced by an online reorganization technique, which runs continuously. Therefore the goal of this DBMS is to optimize the layout of a database for an arbitrary application without requiring any manual tuning.

IBM wildfire and Snappy Data are HTAP systems that build on the scale-out processing engine Spark. Snappy Data uses Gemfire for OLTP requests and the Spark ecosystem for the OLAP requests. Similarly Wildfire has native columnar data storage but it uses the Spark ecosystem for analytical queries, this ensures availability of immediate committed data for analytical processing.

An elastic cloud data storage system(ES^2) was proposed in [CCG+11], which efficiently supports both OLTP and OLAP workloads among the same processing and storage system. The distributed indexing component affirms the distributed data index declaration, such that the efficient processing of ad-hoc queries is facilitated and efficient data retrieval is achieved. Additionally, efficient data loading from various sources, flexible data partitioning scheme, index and parallel scan is provided by the system. The experiments demonstrate the efficiency of ES^2 and yield results stating that ES^2 can provide for most OLAP queries a fresh and consistent snapshot of the data which are simultaneously manipulated by OLTP operations. The benefit of providing distributed indexes and accesses to the data for both OLTP and OLAP queries are confirmed by the yielded results.

In [AIA14], H_2O- an Adaptive Hybrid System was proposed. Ideally, a system should be capable to combine the benefits of all possible storage layouts and execution strategies. Provided a change in workload, it should react to change in real time as workload-specific storage layouts and execution strategies are required for optimal performance. The H_2O system doesn't intend to make any fixed decisions regarding storage layouts and execution strategies, rather the H_2O continuously adapt based on the workload.

The Oracle Database InMemory was defined in [LCC+15] as an option to provide a true dual-format in-memory approach. This DBMS is seamlessly built into the Oracle Data Access Layer, such that the DBMS is allowed to be instantly compatible with all of the rich functionality of the Oracle Database.

BatchDB is an in-memory database engine designed for hybrid OLTP and OLAP workloads [MGBA17]. Good performance, high level data freshness and minimized load balance between the OLTP and OLAP engines over fresh data are achieved by BatchDB.

Google Spanner simply referred to as Spanner[BBB+17], is a globally-distributed data management system that backs hundreds of mission-critical services at Google. It is built on ideas from both the large-scale systems and database communities. Spanner has evolved from being developed initially to work exclusively as a eventually consistent service, until recently it became a relational DBMS with HTAP features. For this various copies of data are maintained in different distributed systems. The concurrency control of Spanner uses the pessimistic locking and timestamps techniques, using Google's True Time technology.

2.1.1.5 Summary on Design of HTAP Systems

As explained above, several HTAP systems like Mem SQL, Peloton, SnappyData, IBM Wildfire, BatchDB, Caldera, OctopusDB and Google Cloud Spanner are available in the market. These HTAP systems have different design and storage methodologies for the OLAP and OLTP workloads. They specifically vary in the data organization and query processing techniques.

To separate the data for analytical processing different options are: snapshots (HyPer), storage views (OctopusDB), multiple versions with delta buffers (SAP HANA) or multiple versions with data containers (Peloton) among others. However, these systems cannot have immediate committed data for the analytical processing [OTT17].

These systems also follow different storage techniques. Systems like SAP HANA use the in-memory columnar processing storage technique. Other systems use PAX layouts which are considered to combine the best of rows and columns (Caldera).

To summarize, the HTAP systems that we have discussed all seek to offer a *"one-size-fits-all"* database system. To accomplish this they have a wide variety in their designs. In the next section we discuss the core requirements or challenges for HTAP systems, and we consider theoretically how the design choices we have presented might specifically affect the two requirements of freshness and isolation.

2.1.2 Challenges of HTAP Systems

The essential goal for an HTAP system is to provide good *performance* for combinations of analytical and transactional workloads. More specifically this means efficient processing of a large number of OLTP clients with high throughput [FK09], while, simultaneously servicing with minimum latency long-running queries for OLAP clients, which in turn can theoretically analyze the most recent operational data. Since there are trade-offs in favoring one type of workload over another, performance per workload is not only a goal but also the *first challenge of HTAP*, which can define as *the challenge of finding the optimal OLTP and OLAP performance (OLAP latency, OLTP throughput) balancing optimizations and resource usage for each workload, at a given number of clients.* These

optimizations tuned for a workload (including, for example, optimal data structures, specialized operators, and others) are one of the essential aspects in database design, supporting the good performance of the DBMS.

Several factors influence the HTAP performance. The work of Psaroudakis et.al [PWM+14] studies some of such factors. In this study authors evaluate the HTAP/mixed workload benchmark CH-benCHmark over two HTAP systems (SAP HANA and HyPer). Based on this they identify 3 key performance-impacting factors when scaling the number of concurrent transactional vs. analytical clients in these databases. The factors are: freshness, flexibility and scheduling. Apart from the factors they establish that there exist trade-offs in configuring them. In the next paragraphs we discuss these factors one by one, after which we discuss the trade-offs and summarize the discussion by presenting 3 challenges related to these trade-offs.

Regarding *freshness* the authors refer to it as a measure for recency of the data consumed by an OLAP query. Traditional batch-processing warehouses allowed OLAP queries to run over relatively stale (or unfresh) data. For example, analysis of frequent purchasing patterns could be scheduled to happen overnight, using the data from the previous day (and not considering the data from the current moment). In contrast, analysis can also be processed over more recent data. For example, real-time analysis of stock data, as they are being updated. In Section 2.2.4.4 we present some metrics that have been proposed in the literature for freshness.

Regarding *flexibility*, authors define it as a measure for how permissive (in contrast to restrictive) is the design for interactivity or expressiveness of transactions and analytical queries. A system that allows UDFs, for example, with ad-hoc queries, would be more flexible than a system which only supports pre-compiled queries. The authors suggest that restrictions in interactivity and expressiveness are usually introduced to enable optimizations.

About *scheduling*, in the paper authors define it as the order or priority in which OLAP and OLTP clients are configured to use the system's resources. When the system reaches saturation, the DBMS might choose to prioritize one type of client over the other.

After establishing these 3 factors, the authors suggest how they could impact one another. In turn we will consider these impacts as 3 additional challenges for HTAP. We discuss them in what follows.

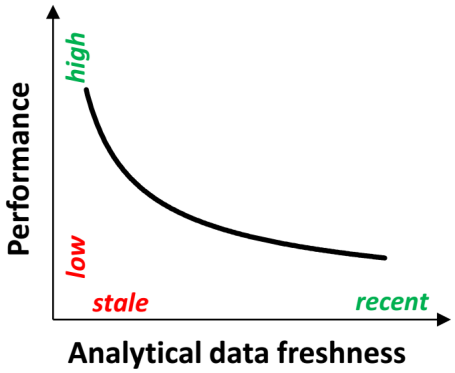
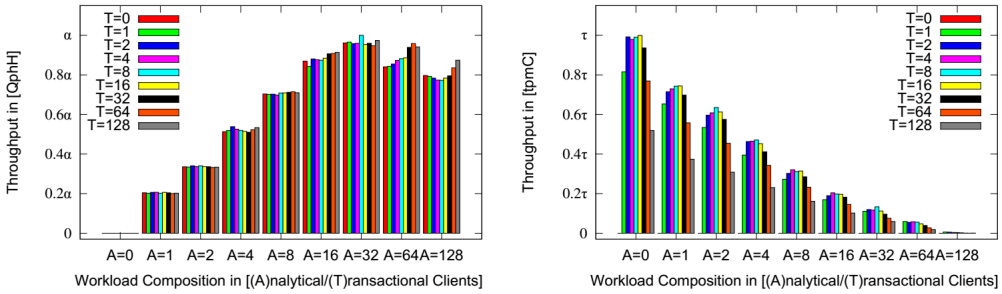
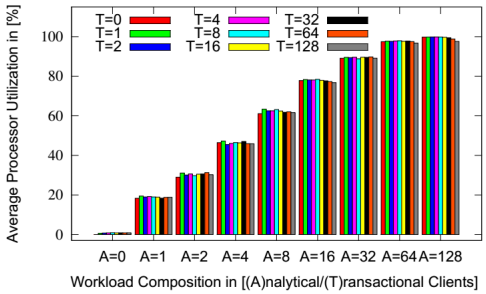


Figure 2.3: Data Freshness[PWM+14]

Increases in the data freshness requirements of OLAP clients reduces the performance, since they must wait for OLTP clients to complete transactions, thus giving information about the most fresh data. Figure 2.3, was proposed by the authors to model, generically, this relationship.

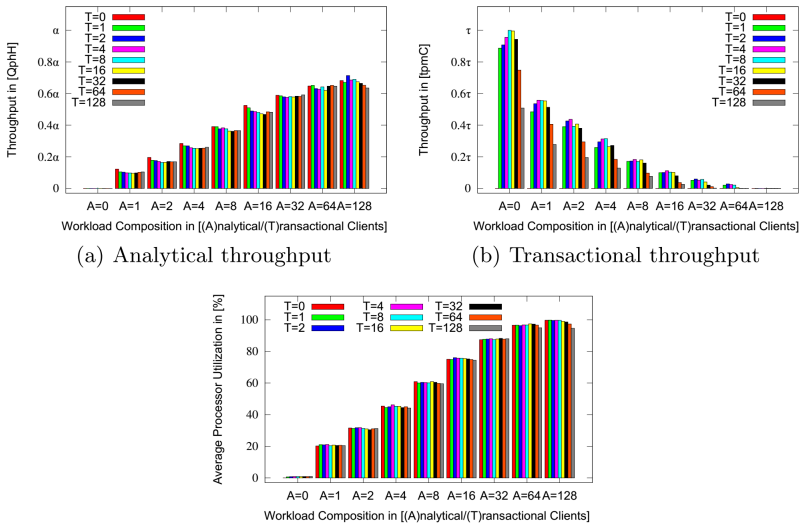


(a) Analytical throughput (b) Transactional throughput



(c) Average CPU utilization

Figure 2.4: Performance of HyPer for low level of analytical data freshness [PWM+14]



(a) Analytical throughput

(b) Transactional throughput

(c) Average CPU utilization

Figure 2.5: Performance of HyPer for intermediate level of analytical data freshness [PWM+14]

By carrying out a performance study using HyPer, authors show that increasing the level of freshness (Figure 2.4 and Figure 2.5) leads to an overall less throughput of OLAP clients, and also to less isolation between clients (which means that with increasing OLAP clients, the performance of OLTP clients is more affected than with less freshness). The last factor also affects the overall performance.

Based on this study we can venture to define the design challenge for HTAP systems related to freshness as *the second challenge of HTAP*. This is *the challenge to serve a given level of OLAP freshness, while isolating OLTP clients from the impact of such freshness requirements.*

In terms of flexibility authors propose that there is also an inverse correlation between flexibility and performance, with more flexibility leading to worsened performance. Authors suggest that this relationship can be modeled, at a high level, as seen in Figure 2.6

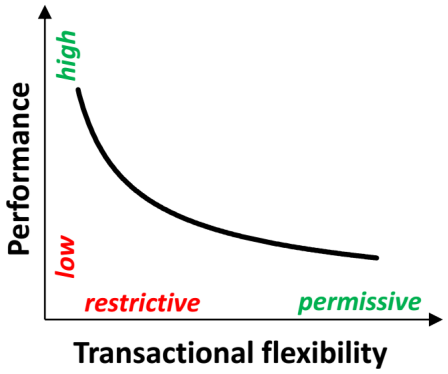


Figure 2.6: Flexbility[PWM+14]

In their study authors do not show a specific case where the impact of flexibility is perceived.

Which gives us the *the third challenge of HTAP, the challenge to provide flexible OLAP queries without deteriorating the overall performance.* It is not clear if this is an HTAP-specific challenge.

While processing both queries at a time, it may saturate the system.There should be balance or scheduling required for the transactional and analytical queries.

In terms of scheduling, the authors emphasize that this is a fundamental task for scaling up mixed workloads. Namely, when receiving a large number of mixed requests, DBMSs have the opportunity of deciding how to manage the number of clients assigned for each workload type. Naturally, this decision is fundamentally related to the first challenge of HTAP. While the first challenge is concerned with optimizations in each workload, and resources given, for a fixed number of clients, scheduling refers specifically to changing the number of clients of each workload type (which ofcourse, also implies different resource usages).

Authors propose that for scheduling and for the first challenge of HTAP, Figure 2.7 can serve as a model: the more resources are spent on workloads of a given type can deteriorate the performance of the other workload type. As a result of this model, we can define the *fourth challenge of HTAP: the isolation challenge of increasing the number of clients of one workload without deteriorating the performance of the other workload type.*

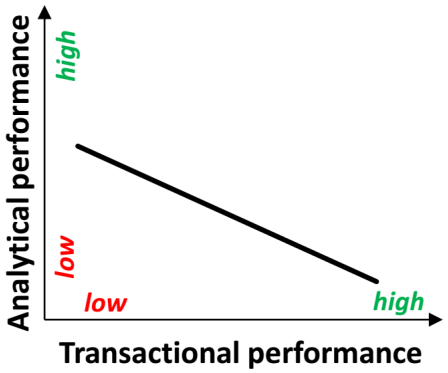
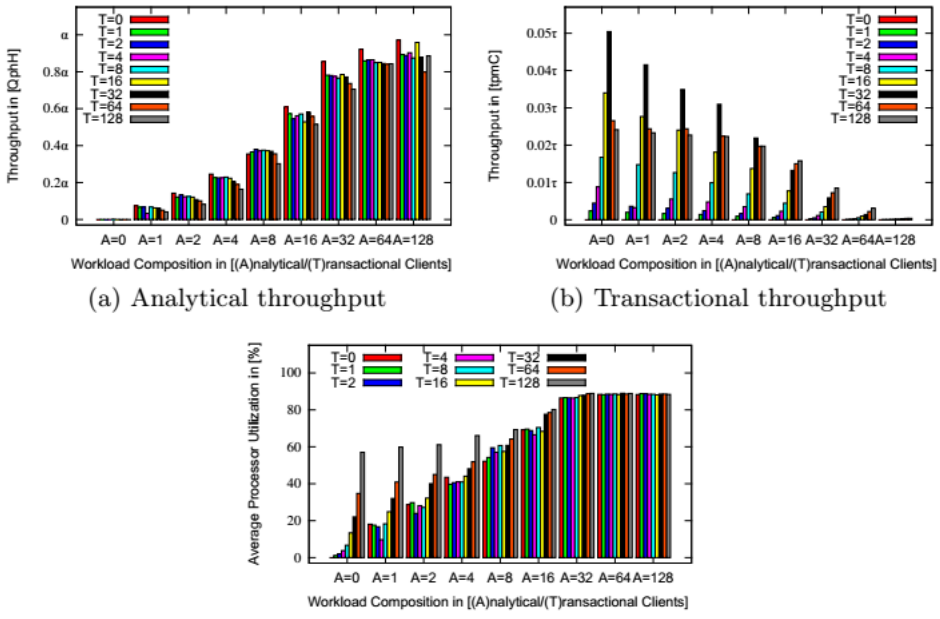


Figure 2.7: Scheduling[PWM+14]

In general, the behavior described for the fourth HTAP challenge produces a "house pattern": when the number of OLAP clients increases, the transactional performance decreases and when the OLTP clients are less the analytical performance is much better than with a higher number of OLTP clients. Figure 2.4 and Figure 2.5 show the result of different combinations of number of clients for the workloads. As stated previously, less freshness lead to better isolation. The more inclination in the slopes, the worse the isolation.

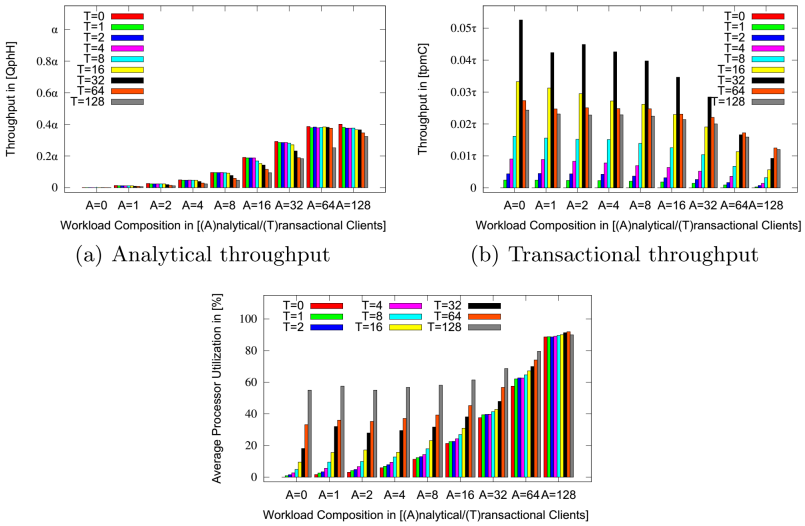


(a) Analytical throughput

(b) Transactional throughput

(c) Average CPU utilization

Figure 2.8: SAP HANA default performance [PWM+14]



(a) Analytical throughput

(b) Transactional throughput

(c) Average CPU utilization

Figure 2.9: SAP HANA performance without intra-query parallelism [PWM+14]

Isolation is specially important to achieve predictable performance of queries, an aspect that is usually one of the expected features from databases. Predictability is an essential user requirement, often formalized as latency bounds in SLAs.

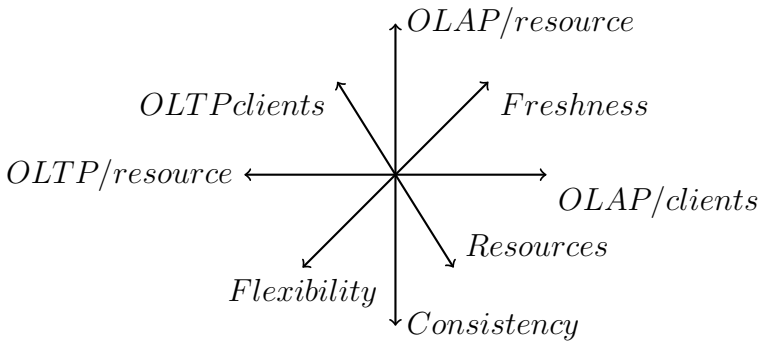
Figure 2.8 and Figure 2.9 report the results of a similar study considering different number of clients using SAP HANA and evaluating different configurations for intra-query parallelism in the OLAP clients. By increasing the parallelism (i.e., effectively giving more resources to OLAP then to OLTP), the performance of the OLAP clients improved, to the detriment of the OLTP clients. This increase in parallelism also produced more CPU utilization and less isolation than for less parallelism. These results reiterate the author's argument that performance per workload/resource assignation (i.e., what we describe as the first challenge) and scheduling (i.e., fourth challenge) have a high impact in HTAP performance.

Apart from these 4 challenges and 3 performance factors, we found in our literature research the mention of other challenges for HTAP design. We mention 2 of them next.

As mentioned previously, the work of Ozcan et.al [OTT17] constitutes one of two tutorials on HTAP systems given at database conferences. This work classifies HTAP systems according to them being a single system or composed by a coupling of systems, they also outline differences in data organization. To conclude the work, they set forward one goal/challenge for HTAP systems based on the definition of Gartner. They call such goal "true HTAP", and it is the goal of having efficient support for OLAP and OLTP *within a single transaction*. To our understanding authors suggest that the design has to consider queries that are a mix of OLAP and OLTP rather than workloads only. To our knowledge supporting efficiently such queries implies a novel HTAP challenge, which we define as the *fifth HTAP challenge*.

Related to this challenge, we would like to mention two implications: First, queries that combine OLTP and OLAP might refer by default to the ultimate freshness, which in turn could be affected by the second challenge of HTAP. Second, if the workload turns out to be composed solely by queries of this type, the analysis for the scheduling might either need fundamental changes or the scaling of clients needs to be re-framed by considering the proportion of OLAP and OLTP sub-queries within transactions.

The final challenge is proposed in the work of Makreshanski et.al, [MGBA17]. In their database presentation paper, they define elasticity as what we can name *the sixth HTAP challenge*: that the DBMS should be able to scale dynamically with changing number of machines (e.g. CPUs) and resources (e.g. disaggregated memory), taking good advantage of the resources provided. While elasticity is important for regular databases, it is specially challenging for HTAP, since studies have shown that these different workloads have a different sensitivity to resource increases, according to scale factors. For example, the study of Sen and Ramachandra [Sen18], reports the effect of increasing the number of CPUs on mixed workloads with mixed clients. For OLAP they find that on small scale factors the performance can benefit more from CPU increases than for OLTP, however on large scale factors the authors report the opposite. Authors also report similar findings for changes in LLC capacity.



To summarize, according to our literature review, we find that there are 6 challenges as suggested in the literature for HTAP systems. These are:

- **Performance per workload**
- **Freshness maintaining recent data**
- **Flexibility tune the freshness**
- **Isolation with changing number of clients**
- **True HTAP OLTP and OLAP within one transaction**
- **Elasticity with changing the number of machines**

To standardize the different systems available in the market, Benchmarking is introduced. The detail about benchmark is explained in the next section.

2.2 Benchmark

A Benchmark is a "standardized problem or a test which serves as a basis for evaluation or comparisons (as for computer system performance)" [Bog12]. In the early 1980's, the world was transforming from end-user transaction, which involved human to human interaction, to an increasing automation of transactions, which lead to more and more human computer interfaces for everyday tasks. One of the first applications was automated teller transaction machine (ATM) which extends from bigger level to smaller level marketing. At that time the on-line transaction processing system industry was intuitive and industries were growing. Today this on-line transaction processing industry has grown to a level which can influence the economy of a country. Huge numbers of vendors are coming up with their services in this area. It is important to know the best among them. To facilitate this requirement, a benchmarking test would be necessary. In the past two decades so many ad-hoc benchmarks for database and transaction processing systems have evolved. The evolution of the benchmarks and the standardization's is clearly summarized in a handbook by J. Gray [Gra91]. Most of the benchmarks created in the early periods were supplied with unclear measures such as transactions per second and query processing performance. These may not be applicable for measuring all the types of enterprise systems. Due to this scenario, many enterprise system vendors came up with their own benchmarks to test their own product. Similarly, SQL vendors have implemented the Wisconsin benchmark and used it to test the performance of their new releases, and with each new machine. On the other hand the Datamation query set which is a DebitCredit transaction benchmark was used to evaluate and compare various relational products. Most of the times the results of the performance tests were not published as the numbers were not so attractive and are not helpful to enhance their markets.

However some third parties used to compare various existing products and used to publish the reviews and results. The losers of these test results used to blame the usability or the credibility of those benchmarks. These situations caused *benchmarking wars* to a significant level. This leads to the development of more standardized benchmarks which are more reliable and acceptable by the most of the elite vendors. These standard benchmarks are developed by some non-profitable organizations and some universities to test performance, price and other important aspects of enterprise applications systems. The Transaction Processing Performance Council (TPC) [Cou17b], The Standard Performance Evaluation Corporation (SPEC) [Cor12] and The Perfect Club (A group of vendors and universities defining benchmarks for the scientific domain, with particular emphasis on parallel or exotic computer architectures) [Gra91] are different organizations which provide standardized benchmarks for enterprise applications.

In the below sections different benchmarks and classifications are discussed.

2.2.1 Benchmark Classification

According to the book by Anja Bog "Benchmarking Transaction and Analytical Processing system" benchmarks can be classified according to their evaluation target, either as

software or hardware benchmarks. [Bog12]. Hardware benchmarks are again divided into two types as component benchmarks and system micro benchmarks.

Hardware benchmarks are useful for testing the system hardware. Component benchmarks test test complete complex components (as the name says), such as a hard drive. However, the micro benchmarks test the micro part of the system such as floating point operation of the CPU.

Software benchmarks, on the other hand, can be divided between application software benchmarks, system software benchmarks and micro-benchmarks. Application software benchmarks evaluate the performance and functionality of applications or business software. System software benchmark compares the software that is at a service application level. Software benchmarks also have Micro benchmarks which compare the alternative algorithms.

The above mentioned benchmarks are used for testing the hardware, software and service applications. None of them are useful to test a database system. A Domain specific benchmark is necessary to test a database system where the performance of such system depends on the existing algorithms rather than on the hardware capacity alone. Among various types of benchmarks discussed above, TPC benchmarks are designed as domain-specific benchmarks. TPC founded in 1988 by group of 34 software and hardware vendors, and governed by Omri Serlin. Among their measurements TPC offer and some of the standard benchmarks used in industry for years to evaluate transaction processing systems. [Gra91].

2.2.2 Transaction Processing System Benchmarks

Transactional processing system benchmarks are specially dedicated to OLTP applications. These benchmark workloads carry only transactional workloads. Usually these study the number of transactions or operations a database can perform in a given interval of time. Each benchmark differs in the way of executing workloads and and the data models assumed. TPC-A , TPC-B, TPC-C, TPC-E and TPC-H benchmarks are the main contributions from TPC . TPC-A and TPC-B are application specific benchmarks. TPC-C and TPC-E are mainly focused on the transaction processing domain.

2.2.2.1 TPC-C

The TPC-C benchmark was developed by TPC on the basis of existing OLTP benchmark in the year 1992. TPC-C benchmark extended the TPC-A and TPC-B benchmarks with real time business's characteristics [Cou17b]. TPC-C considers the real time scenario of wholesale vendors whose business normally carries out on sales territory with its own warehouses. The real challenges to the OLTP system in the above mentioned scenario are, huge number of transactions and the complex database structure. TPC-C benchmark is designed in a way to test whether OLTP system could handle the above mentioned challenges. TPC-C database schema is as shown in the Figure 2.10. It consists of 9 tables with 92 attributes which reflects real time scenario.

One interesting aspect of TPC-C, is that it has a data generator which can be scaled (i.e., the number of data items grow) according to the number of warehouses as shown in the figure (Figure 2.10).

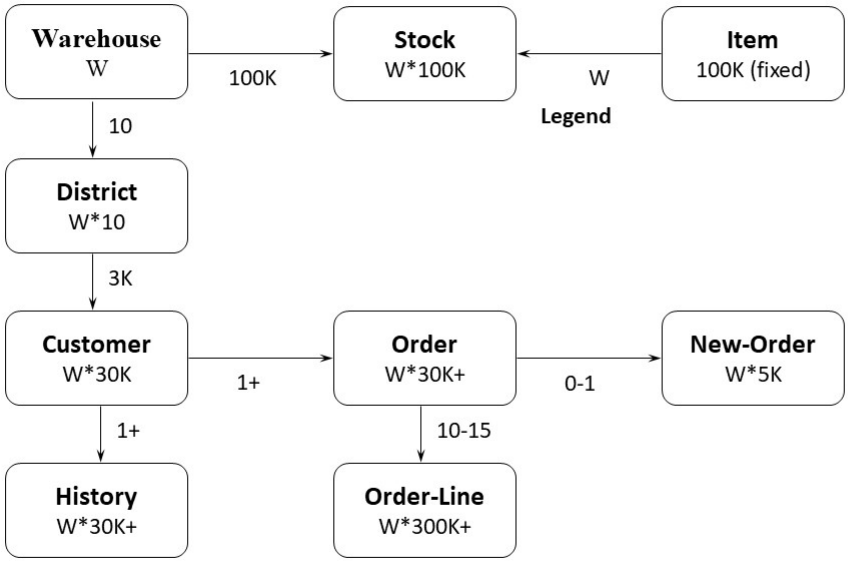


Figure 2.10: TPC-C database schema [Cou17a]

Workload:

TPC-C benchmark uses three read-write (RW) and two read-only (RO) transactions namely

- New-order: enters a new order from a customer
- Payment: updates customer balance to reflect a payment
- Delivery: delivers orders (done as a batch transaction)
- Order-status: retrieves status of customer's most recent order
- Stock-level: monitors warehouse inventory

Each of these transactions in the workload exists in different proportions. New-order and payment transactions hold 45 and 43 percentages respectively. Other transactions like delivery, order-status and stock-level holds four percentages each. TPC-C workloads

require to satisfy ACID property (Atomicity, Consistency, Isolation, and Durability). TPC-C workload is database-intensive with considerable I/O and cache load [BT12].

Metrics

The TPC-C benchmark measures performance and price-performance ratio of a system. Performance is measured as the maximum number of new order transactions per minute (tpmC) that a system can perform in a minimum time period of two hours. Price-performance ratio is measured by sum of costs of hardware, software and supporting worths for three years pricing divided by performance (tpmC).

2.2.2.2 TPC-E

The database benchmark TPC-E was developed and published in 2007 by TPC. TPC-E is a platform independent benchmark. TPC-E considers brokerage enterprise which manages customers, executes customer transactions and interaction of customers with financial market. To handle above scenario, TPC-E database schema groups (Figure 2.11) 33 tables into 4 sets namely brokerage, customer, market and dimension. These tables all together have 133 attributes. It uses a mixed and complex set of 12 transactions.

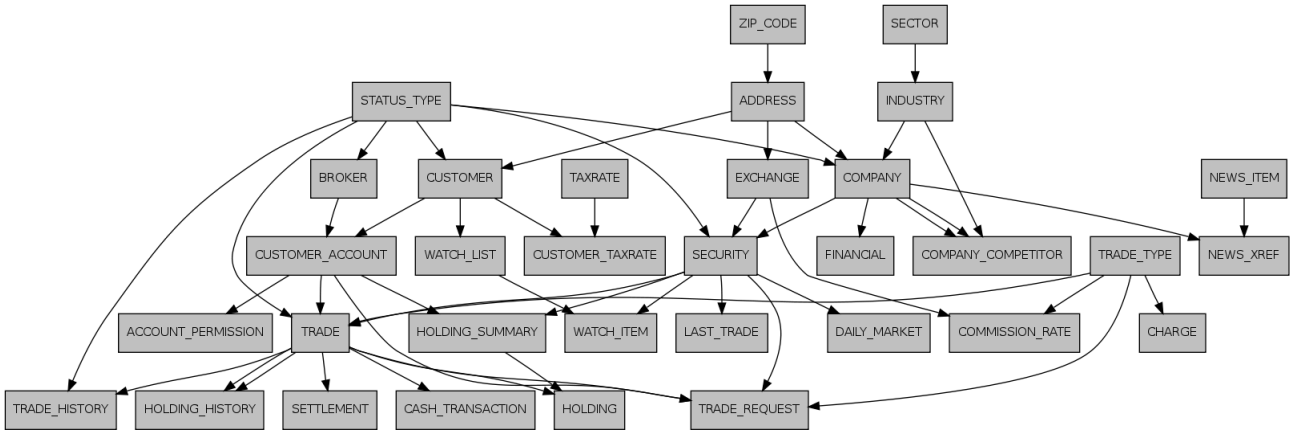


Figure 2.11: TPC-E database schema [BUU17]

TPC-E increased in the complexity of data structure as a result of complex transaction. Compare to TPC-C, TPC-E includes more sophisticated, realistic and referential integrity constraints [CAA+10].

Workload:

TPC-E workload is twice the size of the TPC-C workload. TPC-E has 12 transactions among them six are read-only transactions, 4 are read-write transactions and other two are data maintenance and trade cleanup.

Read-only transactions are namely:

- Broker Volume: Brokerage house's up-to-the minute (potential performance of brokers)
- Customer Position: Retrieves the customer's profile
- Market Watch: It takes care of the process of tracking the current market activity
- Security Detail: Detailed information on a particular security
- Trade Lookup: Information retrieved by a broker or a customer to satisfy their questions regarding set of trades
- Trade Status: Summary of recent trading activity

Read-Write transactions are namely:

- Market Feed: Tracking the current market activity
- Trade Order: Make sales transaction
- Trade Result: Replaced with actual value in any means of trade result
- Trade Update: Update to the set of trade

These transactions in the workload exists in different proportions. Read-only transactions hold 23 percentage and read-write transactions hold 77 percentage. Data-Maintenance transaction runs once a minute. The Trade-Cleanup transaction runs once before starting a benchmark run. Due to more read-only transactions in TPC-E, it is regarded as read intensive benchmark. TPC-E satisfies ACID, in addition it holds check constraint and reference integrity [Cou17b].

Metrics

TPC-E benchmark measures performance and price-performance ratio. Performance is measured as the number of trade result transactions per second that a system can perform in a minimum time period. Price-performance ratio is measured by the sum of the costs of hardware, software and additional support, during three years divided by the resulting performance (tpsE).

2.2.2.3 YCSB Benchmark

Various cloud data serving systems are arriving in the market , which are not based on the relational model. Upcoming Database management systems are interpreted as No SQL database systems, which means they do not share any common data model. The storing of these data is based on key-values-stores which are wide-column database stores. These data stores are different from RDBMS and measuring the performance of these DBMS is difficult. To address these types of problems, Yahoo! Cloud Serving Benchmark

(YCSB) was introduced by Cooper et.al in 2010. YCSB benchmark assumed the simplest data model and defined workloads with different operations. It is a key-valued-stored model which has ten fields of attributes and the workloads of YCSB benchmark are extendable [CST+10]. YCSB benchmark comes with the support of different cloud service DBMS such as Apache HBase , Apache Cassandra, MongoDB, Redis and others.

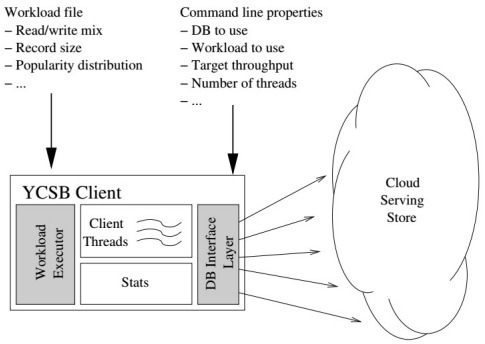


Figure 2.12: YCSB client Architecture [CST+10]

In [CST+10], the authors discussed about YCSB in detail with a focus on performance and elasticity and additionally also work to develop a framework that intend to serve as a tool of cloud system aspects which includes availability and replication in particular. The approaches that can work to extend the framework to these purposes are discussed. A two tier benchmark is proposed to evaluate the performance and scalability of cloud serving systems. The major goal behind extending the benchmark to more number of tiers was to deal with availability and replication of data.

Workload: *Core package*, a core set of workloads to evaluate various aspects of system's performance was developed in [CST+10]. The collection of related workloads is denoted as a package in this framework. A particular combination of read/write operations, distributed requests, data sizes etc. is considered as workload and could also be used for system evaluation at a given point in the performance space. A wide slice of the performance space is examined by a package which contains multiple workloads. YCSB provides it's users to develop their own space in two ways: either by defining a new set of workload parameters or by writing a java code when necessary. Different types of workloads were run in the code by authors such as:

Workload A: It is named as Update heavy, as in this workload 50 percent read and 50 percent of updates.

Workload B: It is named as Read heavy, as in this workload 95 percent read and 5 percent of updates.

Workload E: It is named as Short range, wherein records with ranges upto 100 were used .

Other workloads: Workload C is read only i.e., only 100 percent read and workload D is read latest and considerably returns results that are similar to workload B [CST+10].

2.2.3 Analytical Processing System Benchmarks

Analytical processing system benchmarks are specially dedicated to decision support systems (DSS). These benchmark workloads operate over large volumes of data, complex queries and evaluates database solutions for business decision making. These benchmarks use to measure the performance of OLAP applications systems rather than the performance of a single task. For OLAP system APB-1 was the first standard benchmark of public domain, for example Applix, Hyperion and Oracle. ABP-1 used to determine whether service providers actually offer a minimum set of OLAP functionality to be seen as analytical services. However, the APB-1 does not consider the optimization of query reporting [BBF15].

2.2.3.1 TPC-H

The TPC-H benchmark was developed in 1999 as a decision support benchmark. This benchmark is composed of handling concurrent data modifications and complex ad-hoc queries. TPC-H benchmark queries are designed in a way that the database resembles real time business. This benchmark illustrates decision support systems with large volumes of data, queries with a high degree of complexity to provide solutions to the critical business questions. The TPC-H benchmark database schema has 8 tables and 61 attributes in total as shown in the Figure 2.13.

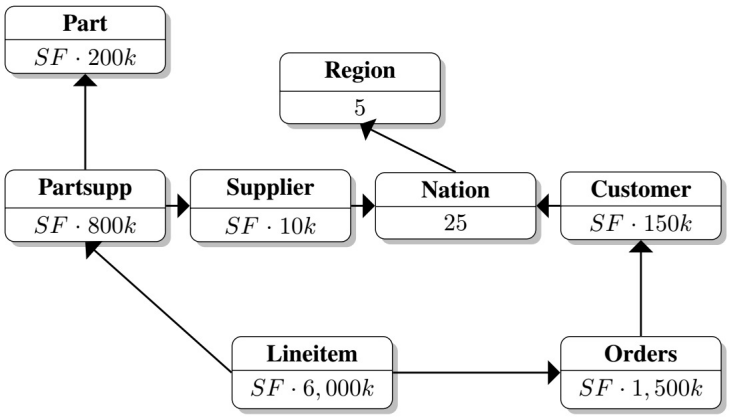


Figure 2.13: TPC-H database schema [FKN11]

Workload:

TPC-H has 22 queries and 2 refreshing functions. The DSS's query optimizer executes these queries in a parallel manner. The two refreshing functions are used to keep the database updated and to remove the old data. The queries are classified and discussed as below:[tpc.org/tpc-h document]

Pricing Summary Report Query (Q1): Reports the quantity of business which was built, shipped, and returned. A sample query is given below:

```
select
        l_returnflag,
        l_linestatus,
        sum(l_quantity) as sum_qty,
        sum(l_extendedprice) as sum_base_price,
        sum(l_extendedprice*(1-l_discount)) as sum_disc_price,
        sum(l_extendedprice*(1-l_discount)*(1+l_tax)) as sum_charge,
        avg(l_quantity) as avg_qty,
        avg(l_extendedprice) as avg_price,
        avg(l_discount) as avg_disc,
        count(*) as count_order
from
        lineitem
where
        l_shipdate <= date '1998-12-01' - interval '[DELTA]' day (3)
group by
        l_returnflag,
        l_linestatus
order by
        l_returnflag,
        l_linestatus;
```

Figure 2.14: Pricing Summary Report Query (Q1)

The output of such a query usually looks as below:

L_RETURNFLAG	L_LINESTATUS	SUM_QTY	SUM_BASE_PRICE	SUM_DISC_PRICE
A	F	37734107.00	56586554400.73	53758257134.87

SUM_CHARGE	AVG_QTY	AVG_PRICE	AVG_DISC	COUNT_ORDER
55909065222.83	25.52	38273.13	.05	1478493

Figure 2.15: Output of pricing Summary Report Query (Q1)

Refreshing functions are used to populate the database. RF1 will add new orders to the database and RF2 will remove the old orders from the database.

Metrics

TPC-H benchmark measures performance and price-performance ratio. Performance is measured as the throughput per second that a system can perform (QphH@Size). Price-performance ratio is measured as the price for the total system divided by the performance ($/QphH@Size). These metrics always calculate against the size of the database, indicated as @Size.

Query	Description
Pricing Summary Report (Q1)	Reports the quantity of business which was built, shipped, and returned
Minimum expensive (Q2)	Search the appropriate suppler in a given region for a given part.
Shipping Priority(Q3)	Retrieves the top ten highest valued unzipped orders
Order Priority Checking (Q4)	Finds out the effectiveness of the order priority system and provides the customer satisfaction rating
Local Supplier Volume (Q5)	Provides the local supplier revenue
Forecasting Revenue Change (Q6)	Predicts the revenue changes with conditions such as “what if”
Volume Shipping Query (Q7)	Determines the value of the goods shipped
National Market Share (Q8)	Determines the change of the market share of given nation for 2 years
Product Type Profit Measure (Q9)	Equate the total profit on a given line of parts from a supplier in a year
Returned Item Reporting (Q10)	Find out the customers who have problem with the products
Important Stock Identification (Q11)	Equate stock from important suppliers
hipping Modes and Order Priority (Q12)	Analyses the remark effect of selecting cheaper modes of shipping
Customer Distribution Query (Q13)	Relation between customers and order
Promotion Effect Query (Q14)	Response of a promotion activity
Top Supplier Query (Q15)	Identifies the top supplier who needs to be rewarded, given more business etc
Parts/Supplier Relationship (Q16)	Identifies whether sufficient number of suppliers who can deliver products with given attributes
Small-Quantity-Order Revenue (Q17)	Determines the loss of average revenue if the orders of small scale are not filled.
Large Volume Customer (Q18)	Ranks customers based on size the ordered quantity
Discounted Revenue (Q19)	Equate the gross discounted revenue generated for the sales of parts those were sold for offered price
Potential Part Promotion (Q20)	Equate parts from the potential suppliers,make available for promotional offers
Suppliers Who Kept Orders Waiting (Q21)	Determines the suppliers who are unable to deliver parts on time
Global Sales Opportunity (Q22)	Finds the locations of customers who may prompted purchase

Table 2.4: Summary of Queries

2.2.3.2 TPC-DS

TPC-DS is a decision support benchmark developed by the TPC in the year 2012. TPC-DS examines a large database with ad-hoc queries, reporting queries and extraction queries. The structure of the database falls under star or snowflakes schema. The data in the database are unevenly balanced among dimensions as like non fact tables. TPC-DS practices with large set of random interchanging query set [PSKL02]. The TPC-DS benchmark was designed intellectually by pulling the essential performance characteristics from the unlikeness operation of DSS. Queries modeled by the TCP-DS benchmark address complex DSS analysis by considering addressable response time. Those queries scripted with parameters that can be parameterized at any point of execution. The parameterization can also be a scenario-based execution. Due to such design, the TCP-DS benchmark could manage to resemble the absolute business analysis environment in a broad manner. TPC-DS schema has 24 tables which includes 7 fact tables and 17 dimension tables. Those 24 tables schema carry 425 attributes which resemble retail goods supplier.

Workload:

TPC-DS workload consists of three distinct disciplines: Database Load, Query Run, and Data Maintenance. TPC-DS workload executes 99 queries called a query run. This runs two times in the order of first database load followed by query run and second data maintenance followed by query run.

Database Load: Database load is primary step of the TPC-DS workload. This carries out hardware and database preparations. Among those preparations some are timed (e.g : loading of base table, creating and validation of constraints, creation of auxiliary data structures and analysis of tables and auxiliary data structures) and some are un-timed (e.g : System preparation, Flat file generation, Permutation of flat file rows, Database creation, Table space creation).

Query run: This model generates a random set of queries from 99 query templates. The query sets are generated by means of query substitution [PS04]. The model can generate an enriched query set by considering 4 factors namely Query class, Schema coverage, Resource utilization and SQL features. Query class in the context of business intelligent systems can be weighed as Reporting class, Ad-Hoc Class, Iterative Class, and Data Mining Class. Schema coverage is the basis for scoping the amount of data that queries will access. The generated query could cover the scope of One fact table, Multiple Fact Tables, Only Dimension Tables, Only Store Sales Channel, Only Catalog Sales Channel or Only Web Sales Channel. The generated query should also considered the different flow of CPU utilization by means of read write operation. As ANSI improved in the quality of querying syntax for DSS system, the workload queries have also to be improved in the form of more complex and concurrent DML queries.

Data Maintenance: Data maintenance plays a vital role in DSS. To resemble the real world mechanism TPC-DS uses ELT (Extract Load Transformation) for data

maintenance. The main argument behind data maintenance is refresh data set. TPC-DS provides the refresh data set in the form of a flat file. ELT mechanism of TPC-DS extracts the data from flat file and loads it to the internal table. The data from the internal table are transfered to fact and dimensional table in their manner.

Metrics

TPC-DS measures the performance (QphDS@SF) and price per performance ($/QphDS@SF). The scaling factor is a number by which the initial database sizes are multiplied to create a larger data set. The performance metrics decided by three scenarios.

- Performance tuning on the Database load, Queries and Data Maintenance
- Use of materialization
- Increasing number of streams

2.2.4 HTAP Benchmarking

In the Section 2.1 the need and the evaluation of HTAP systems is explained. As there is a dramatic growth in HTAP systems, the benchmarking is also in need to be developed. There are standardized and widely used benchmarks addressing either OLTP or OLAP. However, despite the almost decade of HTAP systems, there is no standard HTAP benchmark. With the base of OLTP and OLAP workloads, mixed workloads came into existence. In the following section we are going to look some of the HTAP benchmarks proposed to consider these mixed workloads and the system-specific aspects of HTAP systems.

2.2.4.1 CH-benCHmark

TPC-C benchmark targets an OLTP system using transaction workload. TPC-H benchmark focuses on analytical workload and refresh functions of an OLAP system. TPC-C and TPC-H both can be simply installed on a single database instance and run in parallel, but having different workloads and running on separate data. That's why CH-benCHmark came into existence by executing complex mixed workload of both transaction and analytical types. In [Gar14] Gartner had described that the complex mixed workloads should be designed by considering following challenges in the context of OLAP and OLTP. Namely, continuous data loading, batch (expected) data loading, large number (thousands per day) of standard reports and random unpredictable, ad-hoc query users. Mixed workloads means analytical plus transactional workloads run in parallel on the same tables in single database system. As discussed before CH-benCHmark's mixed workloads drives transactional and analytical workloads from TCP-C and TCP-H respectively.

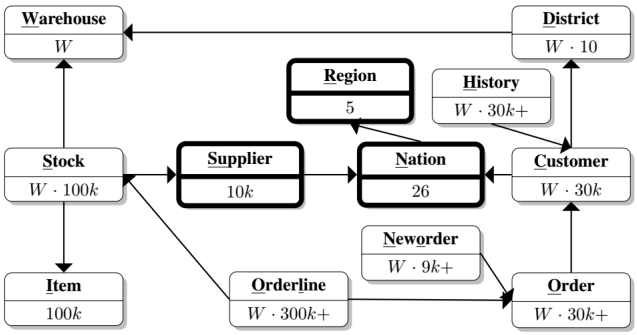


Figure 2.16: CH-benCHmark schema, combining the schema of TPC-H and TPC-C.[FKN11]

Workload:

Five TPC-C business transactions namely, New-order (enter a new order from a customer), Payment (update customer balance to reflect a payment), Delivery (deliver orders), Order-status (retrieve status of customer's most recent order), Stock-level (monitor warehouse inventory) are used without any modification. These transactions are processed on the unchanged TPC-C tables. CH-benCHmark ensures continuous scaling model by increasing the transaction load. An increase in the transaction load acquires increased number of warehouses and also number of terminals. Each terminal generates a limited load due to think times and keying times.

Read-only query suite modeled after TPC-H. CH-benCHmark adapted 22 TPC-H quires and reformulated to match extended TPC-C schema but without any modification in business semantics and syntactical structure. As we know that the TPC-C transaction loads are designed in a way that the database will be updated continuously, so that refresh function is not adapted by CH-benCHmark from TPC-H. As mentioned above the transactional workload continuously increases number of warehouse and terminals that indirectly results in variable database size for analytical workloads. CH-benCHmark workloads are composed of analytical queries only, transactional queries only, or combination of two. OLTP and DS streams are connected to the database system to handle set of transactional and analytical queries respectively. CH-benCHmark's database preserves isolation. Low level of isolation increases the number of concurrence for faster performance and higher level of isolation guarantee the higher quality of results for both analytical and transactional queries. The freshness come into the context because database architecture has a single data set for both workloads.

Metrics:

In the simple model, transaction-per-min-C can be considered as transaction throughput (TpmC) and query-per-hour-H can be considered as analytical throughput (QphH).

Large data volume gives higher transactional throughput but results in less analytical query throughput.

2.2.4.2 HTAPBench

Hyper transactional and analytical benchmark (HTAPbench) is designed to examine the HTAP system, how it could sustain for mixed workloads (transactional and analytical query) without ETL. CH-benCHmark stated that high throughput of OLTP transactions will result in lower analytical query throughput. The HTAP system's goal is to attain the OLAP query performance along with high OLTP throughput. HTAPbench proposed a way to attain a goal by maintaining OLTP throughput always in customer expected interval. This approach ensures stable OLTP throughput and increases the capability of SUT (System under test) to focus on OLAP querying. The HTAP benchmark designed can broadly classified into three stages.1.The populate stage, 2. The warm up stage, 3. Execution stage as shown in the (Figure 2.17) .

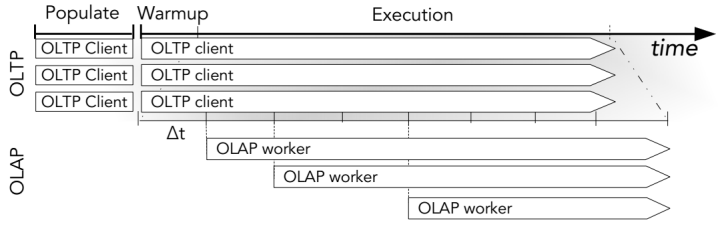


Figure 2.17: HTAPBench execution [CPV+17]

Workload:

HTAPbench chose TPC-C and TPC-H as optimal workload for OLTP and OLAP system. Due to configuration and scalability of workload on the real time scenario database, TPC-C has been chosen. TPC-H data schema is not based on the dataware-house schema and not relay on ETL extraction (similar to the TPC-C schema). Thus HTAPbench workloads are the combination of TPC-C and TPC-H workloads. As discussed in Section 2.2.2.1 increasing in the number of warehouse and the client has direct effect on OLTP throughput (Transactions-per-min-C). In HTAPbench the target transaction per minute is one of the configurable criteria. HTAPbench introduces "Client Balancer" that manages how business workloads are distributed to the SUT which make sure that the OLTP throughput stays within a configured threshold. The business queries of the HTAPbench are TPC-H. The HTAPbench enterprise queries are executed on the data that are fed by the transactional queries which will be keep growing. Fixed or dynamic parameter way of configurations are available to SUT system. The fixed parameterization will make the business queries to search on the full domain. The dynamic parameterization will make a business queries to search on different set of data on each different run. The experimental observation from [CPV+17] stated that " a

variation of up to 77 parentage in result set cardinality was observed in comparison with the previous approach ".

OLTP workloads are randomized(in terms of data access) whereas OLAP workloads are sequential. Likewise, a mixed workload will survey the capacity of the system under test (SUT) to simultaneously schedule random and sequential access patterns. In [CPV+17] experiments, HTAPBench configuration was used across the experiment and SUT can be any intensive system (OLAP or OLTP or hybrid) to check with its own metrics.

Metrics:

In [CPV+17] proposed the new metrics called unified metrics(QpHpW). QpHpWstands for " Queries of type H per Hour per Worker".
QpHpW = (QphH / OLAP workers) @tpmC
Unified metrics is the number of analytical queries executed per OLAP worker regarding a system that maintains the configured tpmC throughput. The Client Balancer module is responsible for monitoring and deciding whether or not to launch additional OLAP workers.

2.2.4.3 Freshness

A time-related quality dimension plays a crucial role to represent the degree of data synchronization between original data and replicas in the processing systems where the data is processed. Several notions have been used to describe this dimension: staleness, age, timeliness, freshness, currency, obsolescence, up-to-dateness [Bou04], etc.

In this section we discuss a selection of work on freshness in data management systems. In (Chapter 4) we discuss freshness measures with more depth.

In various scenarios such as data integration or dynamic web database, the continuous update of data plays a key role. Questions such as the following are very frequent: Whether or not the data is fresh enough with respect to the user expectation? Is there an existence of any stale data during extraction? How much present is the most recent data in the given source?

These questions lead to the concept of data freshness, which consists on evaluating how old the data read is when compared to the data updated.

Data freshness is comprised of a set of quality factors, hence each factor represents some aspects of freshness and each factor can lead to its own metrics. Therefore, freshness can be mostly defined as a quality dimension that can be measured in different ways. For instance, while dealing with huge transactional data such as currency in a bank account, the metrics currency would be more related to data freshness concept whereas while dealing with huge data in an web store to calculate to point the price changes of the items the usage of timeliness metrics would be more appropriate. Below we briefly discuss few such existing metrics of data freshness along with scenarios where these metrics play a key role.

The same quality factor that can be measured with a single specific metric can also be measured with several metrics, just in slightly different ways. For example, we can consider one aspects and different ways of measuring it: The time passed since any change in the source data without being reflected in a materialized view is measured by the *currency factor*. This can be measured as: The percentage of extracted elements that have their values equal to the source related values or simply are up-to-date is measured as freshness rate metric. It can also be measured as: The count of updates that have incurred to a source data from the time of data extraction is measured by the obsolescence metric.

In what follows we summarize briefly some work in data management that either discusses freshness or employs it in system design:

- In this study [Bou04], the authors have analyzed the influencing factors and metrics of data freshness and evaluated them. These factors include features such as the synchronization policy, the type of application and the nature of data.

- In the article [CGM03], the authors discussed the necessity to maintain local copies of remote data sources that are fresh whenever the source data is updated. These studies specify on how a local database can be refreshed to improve its freshness by specifying age and freshness.

  In this paper [CGM03] the authors have defined a theoretical approach for the problem of data freshness on web by discussing various effectiveness policies. Eventually it has been shown that the optimal policies discussed in these articles can significantly improve freshness and age while using real web data. Also, author points out that as the availability of digital information increases, it will be more important to efficiently collect it and in addition a simple data warehouse cannot refresh all its data constantly.

- While inducing the dynamic web content using traditional cache techniques, there incurs the problem of service backlogs since update handling is in the path of service access requires. Therefore it is necessary to dynamically select the materialized views and to maximize performance while the data freshness is maintained in level of acceptance. To address this issue, authors [LR04] introduce an adaptive algorithm where caching and view materialization is combined to allow the decoupling of access server request and update handling using data freshness.

- A typical web database system continuously receives read only queries to generate dynamic web pages [QL07]. The response would be continuous updating write-only response to keep the web data up-to-date. Nevertheless keeping large amount of data updates on time might be highly difficult and hence the [QL07] authors specified response time and staleness of the data as two important metrics to maintain data quality. Further an approach combining these two incomparable performance metrics was proposed as an algorithm to address this issue.

- [NLF05] In this paper authors discussed the information integration process and shed light above an important yet mostly ignored issue i.e., the information quality. In addition, they discussed the necessity to improve the data obsolescence quality metrics. According to them, data quality metrics such as timeliness, accuracy or completeness of data, play a key role in the data quality. Also, an approach to resolve the information overload issue by filtering important information was proposed.

- The process for data quality (DQ) is analyzed in terms of dimension currency that is to be quantified by partly automated and objective quantification techniques [HKK09]. This concept derives two options for DQ quantification namely real world test and estimation of metrics. The results showed that real world test was not practically possible in large data sets and hence the estimated DQ as metric of currency is economical to determine both current and planned DQ level.

- In this paper [GLRG04] the authors works with a goal to introduce C&C adaptive dbms caching. Here the author considers collection of local materialized view which, in turn, consist of collection copies each copies as a cache. The materialized views such as transaction timestamps, self identification and copy staleness are being defined by selection queries. The author further proposed a cache model wherein cache schema can be defined by the local users as a set of local use along with necessary properties of the cache specified by cache constrains.

In the above listing we have described different techniques for extracting fresh data for the real time analysis. Data freshness or Data staleness is a time-related quality dimension which can be measured in various methods. A detailed description of research in freshness and measurement of freshness is described in the below table.

In this section we discussed a selection of work on freshness in data management systems. In (Chapter 4) we discuss these measures with more depth, giving the specific formulas and evaluating them. In the next section (Section 2.2.4.4) we discuss briefly about freshness in HTAP designs.

2.2.4.4 Freshness in HTAP and Benchmarks

Authors Ramamurthy et al. [RDS02] proposed storing the data and replicating it in two different data formats such as row and columnar in a single system. The main advantage of this technique is, to it process the required workload effectively and in addition the can use the resources the resources efficiently. However, the main challenge is to maintain the freshest data and the scalability of the OLAP query performance. To overcome these issues, data replication is one of the novel solution.

Authors Makreshanski et al. [MGBA17] of BatchDB have implemented two types of replication techniques. The primary replication technique handles the OLTP workloads and the secondary replication technique works with updated data which handles OLAP workloads. The two workloads utilize the resources effectively and results in high

Authors	Measurement
Heinrich et al. [HKK09]in 2009	Normalization, Interval scale, Interpret ability, aggregation, adaptively, feasibility
Naumann et.al [NLF05]in 200	Data update rate
Röhm et.al [RBSS02] in 2002	Freshness as parameter in the OLAP queries
Guo et.al [GLRG04] in 2004	Combination of currency and consistency into SQL queries
Labrinidis and Roussopoulos [LR04] in 2004	Proposed an performance and freshness preference
Golab et al. [GJS09] in 2009	Data staleness defined as the time elapsed between present time and timestamps of update tuple.
P.A. Bernstein et al. [BFG+06] in 2006	propose a model that allows user to specify freshness constraints and read out-of-date data within a serialized transaction
Hongfei Guo et al [GLR05] in 2005	Proposed a evaluation on data replication
Cho and Garcia-Molina [CGM03]	To maintain the data fresh used a cache update method
Xiong et al [XHLC08] in 2008	To provide freshness updates

Table 2.6: Overview of Different Freshness Measurement

performance. The OLAP queries always run on the recent snapshot of the data and it can be achieved by scheduling updates in batches from the transactional side. The design of the system solved the issue of high performance and the processing analytical queries on fresh data. However, there is no such measurement technique for freshness or staleness of the data in HTAP available.

Authors Juchang Lee et al. [LMHK+17] proposed a new replication database architecture system known as Asynchronous Parallel Table Replication (ATR). This new architecture allows OLTP transactions in a primary machine and huge OLAP queries as replicas. This schema results in the maximum scalability of OLAP system. In addition, it also avoids the overhead of OLTP processing by freeing the CPU at primary system. Row and column stores are used at primary for OLTP transactions and OLAP queries at replicas respectively. Because of two different primary and replica systems, this architecture cannot analyze real time data. However this ATR architecture allows the user to pre-define the allowed data staleness for individual queries. Hence this method ensures to avoid visible delay in the real time processing of data.

Google Cloud Spanner and its open-source version, CockroachDB, have evolved to become fully-fledged database systems with HTAP features. In them, the concept called stale reads and strong reads are used [BBB+17]. These reads are carried out as lock free transactions. Stale reads ensures that the recent data is caught up for analytical processing by using the past time stamp. Strong reads are set up with a greater timestamps. This helps the strong reads to see the result of previously committed transactions. Strong reads are designed either to wait for the most recent replica is completely caught or it retries for the next near by up-to-date replica.

2.3 Summary

In this chapter, we discussed how HTAP systems are part of an evolutionary path that arises from traditional databases, different types of existing HTAP systems and the challenges faced by these types of HTAP systems. The major challenges that these HTAP systems face are performance, freshness, flexibility, isolation and elasticity. Each of these systems follow different methods to overcome these issues. However, there is some research gap in the evaluation of HTAP system benchmarks. Though there exist mixed workload benchmarks, these benchmarks do not satisfy all the requirements, mainly the freshness challenges. To understand the freshness challenges, we have studied the different freshness metrics provided by various authors.

In the next chapter we describe our prototypical implementation to study freshness metrics.

3. Prototypical Implementation

In this chapter we present the framework and implementation for our study. We organize this presentation as follows:

- We start by presenting our evaluation question in (Section 3.1)
- We describe our evaluation prototype (), basic requirements for connecting to YCSB (Section 3.1.2), and our evaluation setup (Section 3.2).

3.1 Evaluation Questions

As discussed previously, HTAP systems need to tackle several challenges in their design (Section 2.1.2). In evaluating these systems, there still seems to be a lack of a standard benchmark, and a lack of standard tests to compare how the systems face the design challenges. Consequently, we believe that there is a large research gap. To address such gap in our study we focus specifically on the the freshness challenge, and we propose the following core research question:

1. In consideration that data freshness can be a performance expectation: How can we test the freshness provided by databases? What could constitute appropriate metrics, from the literature? How do they compare in compute requirements and in informativeness?

3.1.1 Our prototype

As described above, there are different varieties of "one-size-fits-all" systems available. Among them, OctopusDB [DJ11] is one of the most radical one, which supports both OLAP and OLTP workloads in a single system. OctopusDB is one of the exceptional

HTAP systems in terms of architecture. On the performance bases, OctopousDB is enhanced in terms of data storage system and the adaptive physical layout technique. However, providing freshness for a OLAP query system is still a challenge for it.

Considering the idea of the OctopusDB, Pavlo Shevchenko et al. implemented new HTAP systems called Blinktopus. The brief introduction of the system is given below.

Architecture

This system is mainly based on the log-based storage, it means data stores in the log and different data layouts are used i.e., it creates different Storage Views (SV) such as column SV, row SV and Approximate Query Processing (AQP). The process of storing data in the primary log and creating different physical layouts is similar to the concept used by OctopusDB.

A new feature, orthogonal to our research, is introduced in Blinktopus: AQP. It is one of the significant techniques to handle the interactive queries. The main goal is to process the compact summary data instead of entire data sets. Moreover, to process high speed streaming data AQP is the only feasible method to get interactive response time [CGHJ12]. It focuses on approximation result and fast response time rather than the exact result of the query.

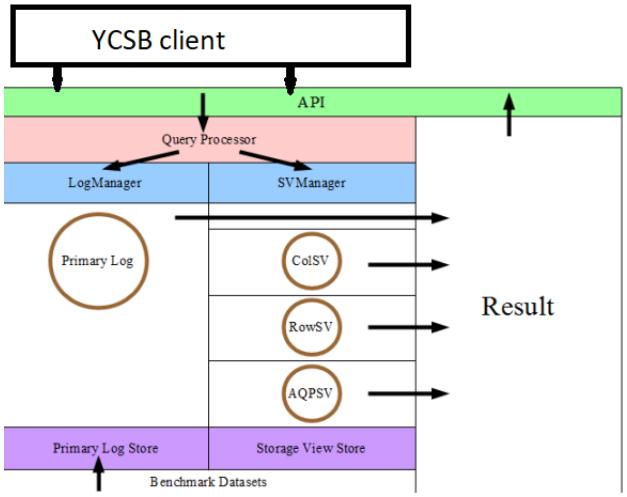


Figure 3.1: Prototypical Database

APQ uses different predefined type of synopsis such as samples, histograms, wavelets, and sketches to summarize the datasets. Each synopsis has its own advantages. Samples can responded to wide range of queries [BC05] and it follows selectivity estimation. Wavelets are used on a large set of data and it is very popular due to one-dimensional computing on the dataset. Histogram, it further divides the dataset into small groups such as buckets, bins and etc.. These small groups further structurally divided according to the query and it provides error estimation. Sketches, this synopsis used for the large and streaming dataset. It follows the hash logarithms. According to datasets, in

Blinktopus, AQP uses the histogram synopsis to provide an interactive response time for the queries.

To be able to use Blinktopus for our work we have modified the general architecture of Blinktopus, so that it connects to the requirements of YCSB Benchmark, and simultaneously, such that it supports a pessimistic concurrency control mechanism and updates between the log and the materialized views with a given refresh rate.

We will be discussing further about requirements of YCSB Benchmark in the section below.

Workflow for using Blinktopus with YCSB -First Blinktopus is compiled and executed, then the dataset is loaded to the database by a YCSB workload. - During the initialization Blinktopus takes as input the refresh rate. - Initially data is stored in the log file called DataLog which stores the recent transactional data. - The operational data is stored in a row or column store. For our research we used the row format, since it matches reasonably well the expectations of YCSB, saving materialization time. -The query processor is the heart of Blinktopus. It directs requests to the respective manager. -Then the workload is compiled and executed, to receive as result the performance of database as a throughput. In the first step of execution, Blinktopus is refreshed and some warm-up time is given before letting YCSB run and return the measurements. - During its executions Blinktopus logs its incoming requests asynchronously to disk, for future usage.

3.1.2 Requirements of YCSB Benchmark

YCSB Benchmark is an open-source specification and program suite that is used for evaluating the retrieval and maintenance capabilities of computer programs. This benchmark is frequently used for the comparision of NoSQL database management system's relative performance. YCSB was built on Java and it possesses extensibility so that the YCSB clients can be modified according to user requirements. This YCSB Client supports to test different databases, therefore we have chosen YCSB Benchmark in our environment as the Benchmark to perform tests over the database.

3.2 Experimental Environment

We have set up Blinktopus and YCSB Benchmark over the Java 8 platform, with system configuration as follows.

- OS: Linux (Ubuntu 16.4)
- Architecture: X86_64
- Width: 64 bits
- Cores per socket: 2

- Model Name: Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz
- RAM: 8GB
- Memory: 500MB

In our evaluation the results for throughput are collected from the YCSB benchmarking tool, the results from the freshness measures are calculated by using a log of the requests given to Blinktopus.

3.3 Summary

In this section we provide an overview of our implementation along our core evaluation question, the workflow and the experimental setup that we have used. In further chapters of this thesis, we will be discussing about the results of the experiments and how we have answered our core evaluation question.

4. The Evaluation of Freshness

In this chapter we present the results of our evaluation of the different freshness metrics as used over runs of the YCSB benchmark and our replica-based HTAP prototype, Blinktopus. We structure this chapter as follows:

- **Evaluation Question:** We start the chapter by recapitulating the research question that motivates our study for the chapter (Section 4.1).
- **Freshness Metrics:** We detail the freshness metrics that we considered, based on our study of the literature, giving the specific formulas used (Section 4.2).
- **Freshness Measures in a Database Benchmark:** We present the results of our evaluation in measuring the selected metrics over executions of YCSB with our prototype (Section 4.3. We analyze our results by comparing the informativeness of the metrics (i.e., how their averages correlate to the throughput) and their computational requirements). We structure our findings in a list of observations, providing a recommendation of the freshness metrics that should be considered for inclusion into HTAP benchmarks. Additionally we propose and test a novel measure that we considered pertinent and missing from the literature Section 4.3.8.
- **Summary:** We summarize our findings (Section 4.4).

4.1 Evaluation Questions

In this chapter we set to answer to the following research question:

1. In consideration that data freshness can be a performance expectation: How can we test the freshness provided by databases? What could constitute appropriate metrics, from the literature? How do they compare in compute requirements and in informativeness?

To this end we consider an offline evaluation of freshness, whereby we execute the YCSB workload and track the metrics by writing to a file the complete execution of the workload. Thus, generated results are analyzed for their update time with respect to reads, for understanding how the performance of a workload varies between some selected time intervals.

For the task of analyzing freshness in an HTAP system we are using *Blinktopus* which was built based on the architecture of Octopus DB [DJ11]. Here the transactional data is saved in an append-only log format. This data, so that it can be accessed by OLAP operations, is periodically added to a snapshot in the shape of a materialized view (i.e. row or column store). Such replica is updated every 3, 30 and 300 second intervals, in our tests.

We employ workloads that update the data present in the system (i.e., the OLTP replica), while simultaneously performing reads (i.e., from the relatively out-of-date OLAP replica). Next to the freshness metrics we report the throughput of the workload. Our observations regarding the last aspect are in line with previous studies showing that the lower freshness leads to higher throughput.

In the presentation of our results, we study measures over given windows between two updates to the read replica (i.e., refresh operations). This implies in the first place the selection of the windows to discuss. Our approach was to select a pre-defined set of windows for all measures, we took special care to repeat this set when we compared measures (e.g. *Freshness Rate* and *Freshness Index*). For representativeness, we also selected to feature results for some windows with a lower number of operations than average in the run. We made such decision for *Freshness Index, Freshness Rate, Absolute Timeliness and Currency Alternative*. In general this should simply portray more scenarios in our study, and should not change our observations in any way.

For the sake of complete disclosure, we include the statistics on the operation distribution across refresh windows in Table 4.1. We show the averages, for which generally 95% (i.e. the number of reads) should match the count of read sequences we report. For 300 seconds we observe a skew of 0 and a very low kurtosis, indicating an almost even distribution with relatively frequent but modestly sized variations. For 3 and 30 seconds we note negative skews indicating that a relatively low number of windows departs from the average with a lower load, and a high kurtosis suggesting that these departures are marked (i.e., that there are few windows with much less operations than average). Hence we feature some results for these cases.

Statistic	**3 sec**	**30 sec**	**300 sec**
Average	5263.16	47796.65	492309
Variance	1369612.41	141221103.1	2942200833
Skew	-2.36	-2.60	0
Kurtosis	5.35	8.28	-6

Table 4.1: Statistics for the distribution of operations across refresh windows in our evaluation

We calculate various freshness metrics using their respective formulas, which we discuss in the coming sections.

4.2 Freshness Metrics

In our work we propose to analyze various freshness metrics, as they concern to evaluating HTAP systems. To this end we devote this section by introducing our compilation of freshness metrics, which we have derived from various studies for determining the freshness of data.

All the formulas that we discuss in this section are based on a small set of measurements, which we can define as follows:

- t_i: Any time point.
- r_{tx}: Any given read transaction.
- t_u: Timestamp in which we updated the complete OLAP MV (materialized view)
- t_{maxtx}: Timestamp of the latest transaction in the OLAP MV
- t_q: Timestamp in which we query for a specific item
- t_{lu}: Timestamp of the last update of the specific item, right before t_q, in the OLTP store (i.e., for Blinktopus, the log)
- t_0: Timestamp of the first update to an item (in its complete history).
- t_{maxi}: Given item i, this is the the max transaction timestamp (i.e, the last update) that the item has before t_u.
- N_u: Number of updates of a given item, up until a given timestamp.
- Obs: Number of transactions between t_q and t_u.
- n_u: Number of items updated since last refresh (i.e., number of items that have not been updated yet in the read replica).
- nmv: Number of items in a read replica or materialized view.
- nmv_{nu}: Number of items in a read replica or materialized view that have not been updated(nmv-n_u).

4.2.1 Two Alternatives for Currency

The first metric we consider is *Currency*. In [BWPT98], the authors have defined currency as a function of various factors such as: delivery time (i.e., when the information product is delivered to the customer), input time (i.e., when the data unit is obtained) and age (i.e., how old is the data unit when it is received).

In essence, currency of a read transaction can be calculated as the difference between extract and read time of the data. The more the currency, the less fresh the data is.

Formula: Currency (r_{tx}) = t_q-t_u. [CGM00], [CGM03], [BSM03], [ES07]

In [BWPT98] the *currency* is interpreted, in an alternative way, as an updated currency measure that captures the time of the last update of the item before t_u.

Formula: $Currency_{alternative}$ (r_{tx}) = t_q-t_u + Age (item)

With *Age* defined as follows:

$$\text{Age (item)} = t_u\text{-}t_{maxi}. \quad \text{[Bou04]}$$

This alternative measure of currency has also been named as *Timeliness*, by other authors. This concept of timeliness [Bou04] [GLR05] for a read transaction is measured as the difference between the time of the query, and the time in which the item was updated in the view, before t_u (i.e., t_{maxi}).

Formula: Timeliness (r_{tx}) = t_q-t_{maxi} (this is interpreted from [Bou04])

From the two previous definitions (currency and currency alternative) we see that the first one considers a measure for read transactions, but without a focus on the individual items. Instead, this metric focuses on the difference between the read and update times for the system as a whole. In contrast, the second measure provides a more fine-grained measure for the freshness of a read transaction, encompassing the last update of the item being read before the last update to the read replica.

4.2.2 Freshness, Staleness and Absolute Freshness

As discussed earlier, *Freshness* is conceptually a measure that gauges if the data in the read replica is up-to-date or not. In contrast, its complementary measurement *Staleness* conceptually checks how outdated is an item in the read replica.

The *Freshness* for a read transaction can be measured as the maximum timestamp of any tuple in the materialized view. Following their complementary natures *Staleness* can be measured as the difference between current time and the freshness of the read transaction.

Formula: Freshness(r_{tx}) = t_{maxtx}. [GJS09] [QL07]

Formula: Staleness (r_{tx}) = t_q-freshness(r_{tx}) [GJS09]

The absolute version of each of these measures can be calculated (per read transaction) by comparing the item in the database with the real time data, and verifying if the data

has gone through any update [CGM03] [CGM00] [BSM03]. In the primary case, that is if the item has been updated after t_u, then the read transaction is absolutely stale (i.e., it returns "0") else, it is absolutely fresh, wherein "1" is returned.

Formula: Absolute Freshness (r_{tx}) = 1 if fresh at t_q; or 0 if stale at t_q.

This measure is applied to each read transaction, and then a counter can collect the number of absolutely fresh or absolutely stale reads in a workload, to provide an understanding of the proportions w.r.t to the total number of reads.

It is important to note that the definitions for staleness and freshness that we have presented thus far focus only on the table as a whole, and are not predicated on individual update times of items.

4.2.3 Freshness Rate and Freshness Index

Two metrics have been proposed to gauge how fresh a read replica is: the *Freshness Rate* and the *Freshness Index*.

The *Freshness rate* is measured as the percentage of items in a read replica or materialized view that are not updated after t_u. That is, the total percentage of items which remain non-updated after there has been a successful update in the OLAP materialized view. This measurement requires to keep track of every updated item since t_u. [Bou04] [CGM03] [CGM00] [GJS09] [ES07] [QL07] [CGM99]

Formula: Freshness Rate(t_i) = nmv_{nu}/nmv.

[NLF05] had referred to the *Freshness Index* as a measure of freshness of the data at a given time. This index measure reflects on how much deviation exits between the up-to-date data and the data in the database. Intuitively derived a freshness index of "1" states that the data or items are up-to-date, whereas a derived index of "0" resembles "infinitely" outdated data.

Formula: Freshness Index(t_i) = t_{maxtx}/t_{lu}

It is important to notice that this measure compares the maximum timestamp in the read replica with the last update of a specific item; as a result we can consider it an item-based measure. We deem that the same can be said for the freshness rate, which though it considers number of items alone, it must count the precise number of items actually updated in the write replica.

4.2.4 Obsolescence

Authors have also proposed *Obsolescence*, defined as age in caching systems, i.e., the count of updates performed on an object in a remote server starting from when it was cached. This measure can be distinguished from others by being time independent, since it does not consider the time but focuses on the transaction count.

In this metric the count of updates that have occurred in the source starting from the extraction time is measured. This measurement is possible from delta files or source logs or even using change detection techniques.

Formula: Obsolescence(r_{tx}) = Obs

4.2.5 Absolute Timeliness

An absolute measure for timeliness has been proposed [BWPT98], wherein the *Absolute Timeliness* of a read transaction is interpreted with regards to volatility. In turn volatility is a measure that models how long an item remains as valid. It can be calculated, on the fly, as the time difference between the last two updates.

Formula: Absolute Timeliness (r_{tx}) = max {1-{currency/volatility},0}

In this formula currency is calculated as usual (i.e., t_q-t_u). The resulting formula returns "0" if the data is unacceptable and "1" if the standard of high timeliness is met. The measurement only needs to keep t_u and the volatility of items.

In addition to this formula, authors have proposed to enhance it with parameters that enable to control the scale of the measure (i.e., by elevating the resulting measure to the power of a parameter s).

Other authors [LMTdU08] have also discussed about the same measure of absolute timeliness, under the term *Up-to-dateness*.

A similar version of this measure, in the context of age metrics, was discussed by Hinrichs et. al. [Hin02]. Authors propose that this metric of timeliness could deliver an indication on whether or not the value of an item has faced any changes in the real world starting from its time of acquisition and storage within the system. If this measurement has mean item update time as "0", then it infers that the item value never turns outdated, whereas a result of "1" refers to an up-to-date attribute value. In contrast, if the item age is "0" then it is outdated.

4.2.6 Three more measures for Staleness, based on the items update history

In addition to the measures presented thus far, authors have also proposed different approaches to model the staleness of a read transaction over an item [OC12]. Unlike other approaches, where the system either had complete access to present and past information, or only to past one, the formulations that we discuss in this section combine data from the live system and only aggregates of the past.

Specifically authors proposed 3 approaches for estimating staleness: *Enhanced Averaging*, *Shifting Window* and *Exponential Smoothing*, leading to 3 different formulas and calculation methods for this measure.

Through *Enhanced Averaging* staleness is calculated by considering the last update of the item (before t_u), the first update of the item, and the number of updates that have happened in between (N_u, not including results for after t_u).

Formula: $Staleness_{EnhancedAveraging}(item, t_i) = t_i\text{-}\ t_{lu} + \frac{t_{lu}-t_0}{N_u}$

This measure estimates that the staleness is the difference between t_i and the predicted next update. For calculating this prediction, the method assumes a uniform distribution between the first and last updates. When the difference is negative the model outputs a staleness of 0. Likewise when an item has not been updated since its creation.

A possible limitation with this approach is the assumption of a uniform distribution in the past update times, which is used to predict the next update. To solve this authors propose the use of a shifting window that goes beyond t_u.

Formula: $Staleness_{ShiftingWindow}(item, t_i) = t_i\text{-}\ t_{lu} + \frac{t_i-t_{windowstart}}{N_{currwindow}}$ if $N_{currwindow}$>=3.

Where $N_{currwindow}$ refers to the number of updates in the current window. When this number is not greater than 3, then authors replace the fractional component by the inverse of the update rate of the previous window.

Another method proposed is the exponential smoothing. Here an average linear weighted combination of updates (Sav_{ti}) is proposed to model data changes through a recursive formula: Sav_{ti}= $(1-\alpha)\ Sav_{ti-1} + \alpha * numberofupdatesinpreviouswindow$. Then, an online method for determining if a staleness value should be given is used.

4.2.7 Summary

In Table 4.2 we summarize the 12 freshness measures that we have discussed in this section. We make a distinction between metrics for which we expect a negative or positive correlation w.r.t the throughput for OLAP operations. Values that measure freshness are expected to be negatively correlated (i.e., the more freshness, the lower the throughput), while values that track staleness are expected to behave in the opposite manner.

We also distinguish global vs. item-based measures, considering whether the measure is dependent on item-specific features.

Finally we also highlight the unit of the measures. Units can be either time-specific (e.g. nanoseconds), boolean (for which an aggregate might be necessary) and counts of transactions.

Measure	**Unit**	**Global or Item-based**	**Formula**	**Expected Correlation with Throughput**
Currency(r_{tx})	Time	Global	t_q-t_u	Positive Correlation
$Currency_{alternative}(r_{tx})$ or Timeliness(r_{tx})	Time	Item-based	t_q-t_{maxi}	Positive Correlation
Freshness(r_{tx})	Time	Global	t_{maxtx}	No Correlation
Staleness(r_{tx})	Time	Global	t_q-freshness(r_{tx})	Positive Correlation
Absolute Freshness (r_{tx})	Boolean	Item-based	1 if fresh at t_q, 0 otherwise	Negative Correlation
Freshness Rate(t_i)	Percentage	Item-based	nmv_{nu}/nmv	Negative Correlation
Freshness Index(t_i)	Percentage	Item-based	t_{maxtx}/t_{lu}	Negative Correlation
Obsolescence(r_{tx})	Count of Transactions	Global	Number of transactions between t_q, t_u	Positive Correlation
Absolute Timeliness(r_{tx})	Boolean	Item-based	max{1-{currency/volatility},0}	Negative Correlation
$Staleness_{EnhancedAveraging}(item, t_i)$	Time	Item-based	t_i- t_{lu} + $\frac{t_{lu}-t_0}{N_u}$	Positive Correlation
$Staleness_{ShiftingWindow}(item, t_i)$	Time	Item-based	t_i- t_{lu} + $\frac{t_i-t_{windowstart}}{N_{currwindow}}$	Positive Correlation
$Staleness_{ExponentialSmoothing}(item, t_i)$	Time	Item-based	Update mechanisms	Positive Correlation

Table 4.2: Summary of Freshness Measures Evaluated

4.3 Freshness Measures in a Database Benchmark

After establishing, in the previous section, the different freshness measures that we will consider, we present in this section our measurements of them over executions of the YCSB benchmark using our HTAP prototype, Blinktopus.

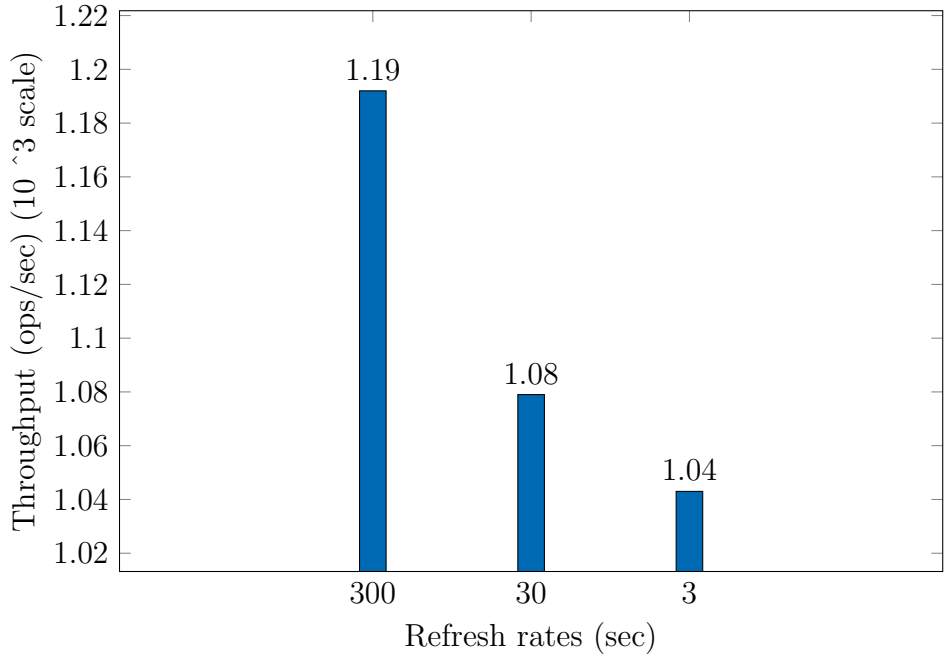


Figure 4.1: Throughput for Blinktopus on YCSB benchmark, workload B, at different refresh rates

In Figure 4.1 we present the results for the throughput for workload B, which consists of a mix of 95% reads and 5% updates, in 50000 operations over 4000 records. The distribution of operation to records followed a default Zipfian distribution, with 80% of cold and 20% of hot data.

In our setup, as explained in Chapter 3, we forward all read requests to a replica in a row layout. Write requests, on the other hand, are appended to a log. We define as *refresh rate* the interval in seconds for updating our read replica with the latest information from the logs. This is an operation that is performed in a consistent manner, through pessimistic locking. This *refresh rate* can be considered similar to a global bounded staleness for read requests. Accordingly, a lower refresh rate (e.g. once every 300 seconds) is associated with a lower freshness, and a higher refresh rate (e.g. once every 3 seconds) is associated with a higher freshness.

Our motivation for testing across different refresh rates is that through this we seek to understand how well can the different measurements proposed be used to distinguish the scenarios.

In our results, in Figure 4.1, we can see that similar to expectations ([BPZ11, PWM+14]) more freshness (i.e., refreshing more frequently the read replica) leads to lower throughput. From this execution we conduct a profile where we log all requests. Next we analyze this log, in order to extract the different measures. In what follows we present and discuss our results for the measures that we have pre-defined.

4.3.1 Two Alternatives for Currency

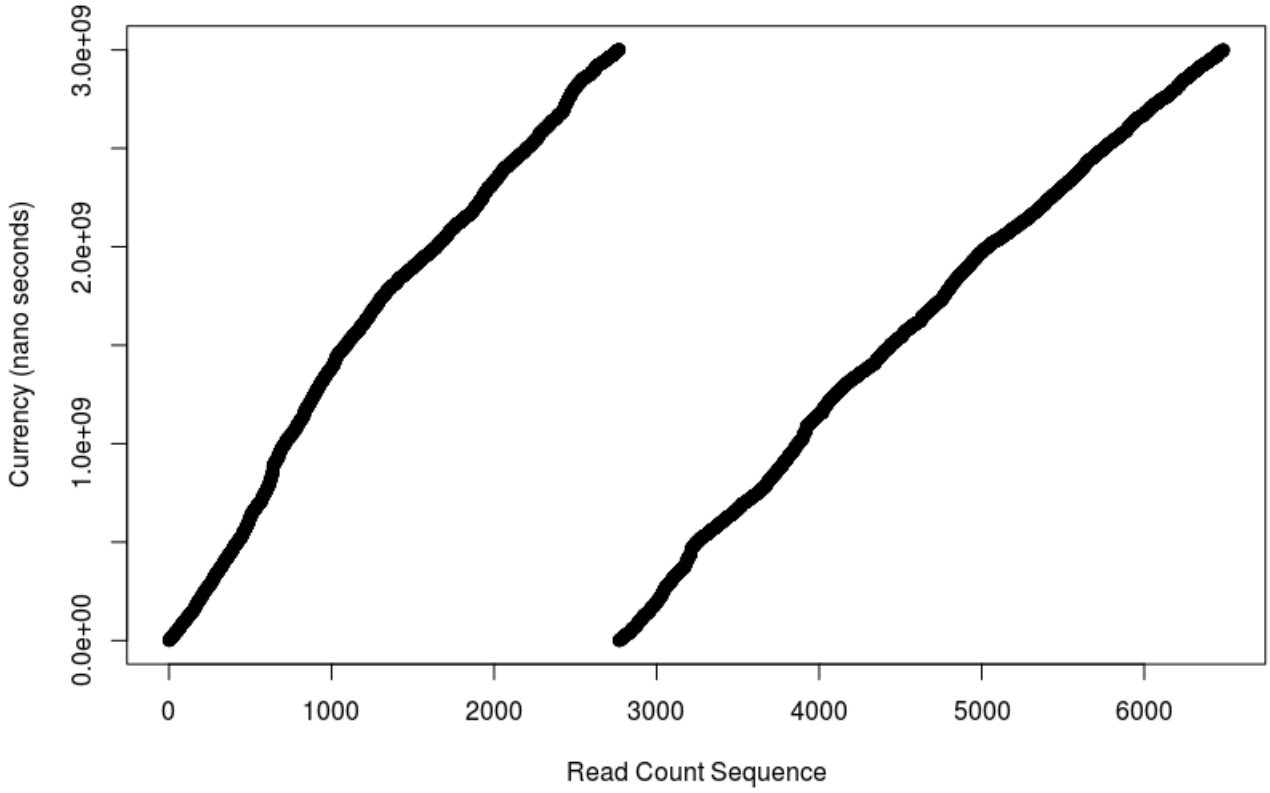


Figure 4.2: Currency at 3 sec for Workload B. Two windows between updates to read replica.

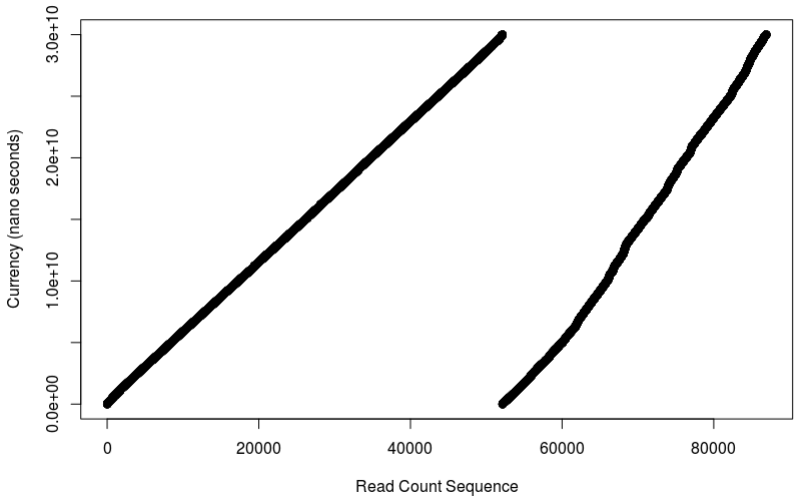


Figure 4.3: Currency at 30 sec for Workload B. Two windows between updates to read replica.

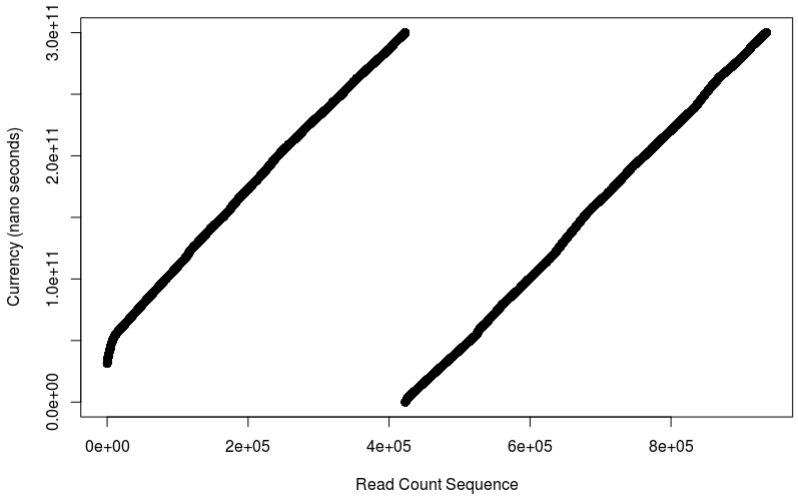


Figure 4.4: Currency at 300 sec for Workload B. Two windows between updates to read replica.

The evaluation results of the freshness factor of *currency* are presented in this section. For this we consider the two alternative formulations that we discussed in previous sections.

Currency is one of the simplest of metrics in terms of measurement, among all other metrics. It is measured by the time elapsed between the timestamp of the specific item

and the update timestamp to the read replica. Higher values of this metric clearly indicate a delay in the read replica, and hence the possible existence of stale reads or less fresh data.

In Figure 4.2, Figure 4.3 and Figure 4.4 we present our results for the currency metric. These figures are plotted for two windows. These number of reads in these two windows is consistent with the observed throughput. The plots are found to be very linear, similar to obsolescence, which is a result of the linear formulation that only depends on t_u and t_q. In addition, as expected the plots exhibit increasing trends with time.

In terms of its capability to compare the 3 different scenarios for refresh rates, we see that, in the nanoseconds in the Y-axis, the measure captures well the growth in the order of magnitudes across the different refresh rates.

However, since the metric is predicated on time, and not on the number of transactions taking place, it provides only high level information and it is difficult to assess the real freshness at a transaction level.

In Figure 4.5, Figure 4.6 and Figure 4.7 we show our results for the alternative version of currency.

Unlike the previous measurement, here each individual read transaction considers the maximum update time before t_u for the item it is reading.

As a result of this change we can see two important differences:

- First, the line plotted is displaced to higher points on the y-axis (since t_{maxi} is smaller than t_u).

- Second, this plot shows more fine-grained results, with items that were updated more in the past (before t_u) moving higher in the y-axis, and items that have been updated more recently (but before t_u) moving lower in the y-axis. Because of this some aspects related possibly to the hotness or coldness of data are visible, specially in Figure 4.7, with clearly visible demarcations between items updated more in the past (higher in the y-axis) and items updated more recently (lower in the y-axis). In addition, densification is the trend as the window length grows.

Apart from these distinctive features, these plots show a similar behavior to those of the previous currency measure: they show a linearly increasing trend, which is consistent per item; across refresh rates they show growths in the orders of magnitudes. In spite of this, this measure suffers from similar limitations to that of the previous currency metric: it does not indicate the proportion of reads which are not fresh. By only employing the last transaction time of the item, it does not model sufficiently the volatility of the item.

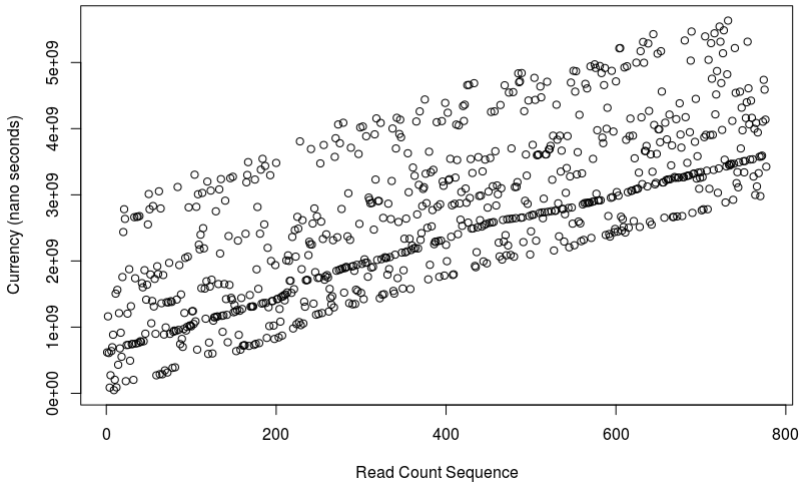


Figure 4.5: Currency (alternative) at 3 sec for Workload B. Single window between updates to read replica.

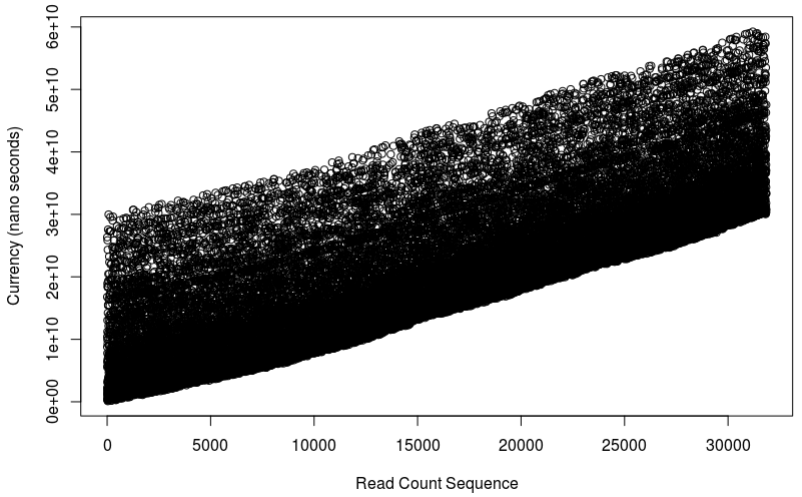


Figure 4.6: Currency (alternative) at 30 sec for Workload B. Single window between updates to read replica.

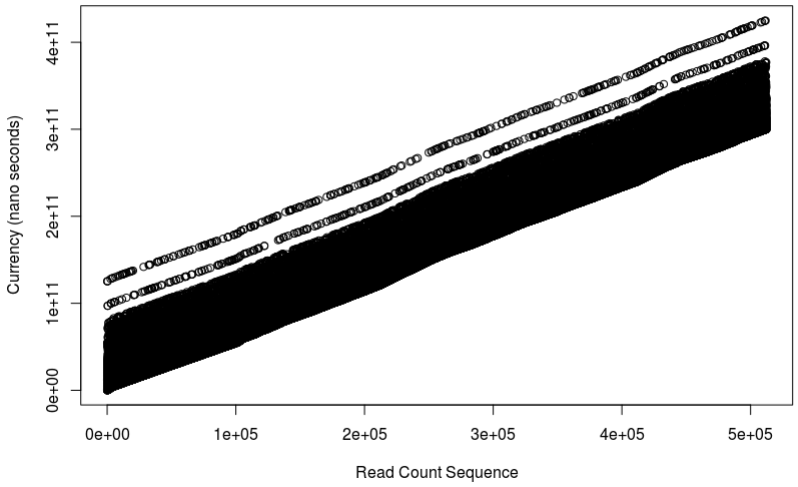


Figure 4.7: Currency (alternative) at 300 sec for Workload B. Single window between updates to read replica.

4.3.2 Freshness, Staleness and Absolute Freshness

In this section we discuss our results for *freshness*, *staleness* and *absolute freshness*. In Figure 4.8, Figure 4.9 and Figure 4.10 we show an example of a measurement of freshness. Evidently, by only considering t_{maxtx} this measure gives as output a constant value throughout the whole window. As a result it is not informative and does not give insights about how the three cases differ.

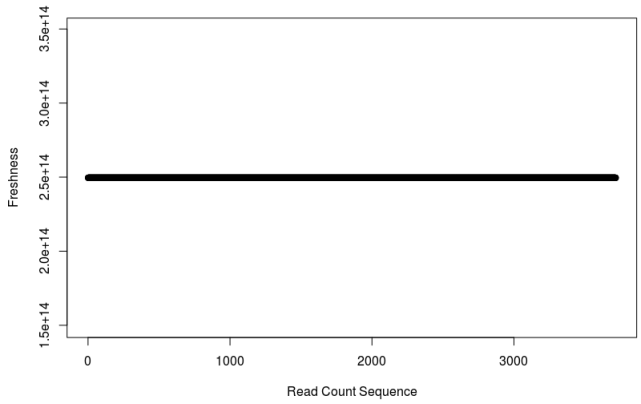


Figure 4.8: Freshness at 3 sec for Workload B. Single window between updates to read replica.

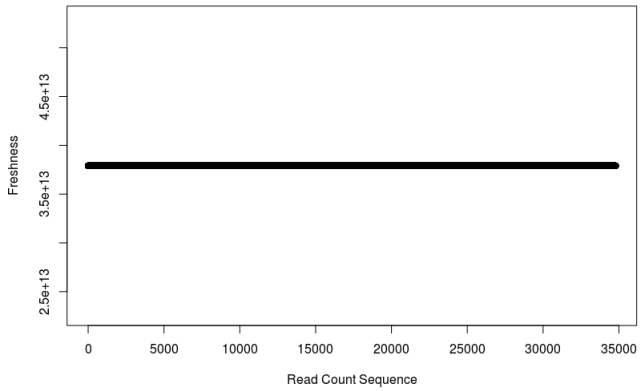


Figure 4.9: Freshness at 30 sec for Workload B. Single window between updates to read replica.

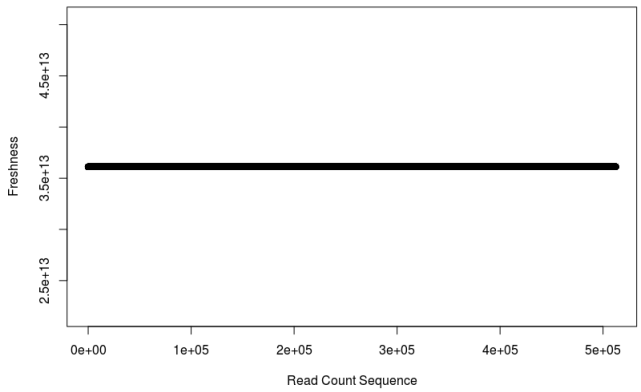


Figure 4.10: Freshness at 300 sec for Workload B. Single window between updates to read replica.

In Figure 4.11, Figure 4.12 and Figure 4.13 we present our results for the staleness on the 3 refresh rates. This measurement is almost equivalent to currency, save for the fact that the offset is not on t_u but on t_{maxtx}. As a consequence, the same limitations from that measurement apply for this definition of staleness (i.e., it fails to inform on the freshness of individual items in consideration of their volatility, it does not aggregate to give insights about the freshness of reads as a whole).

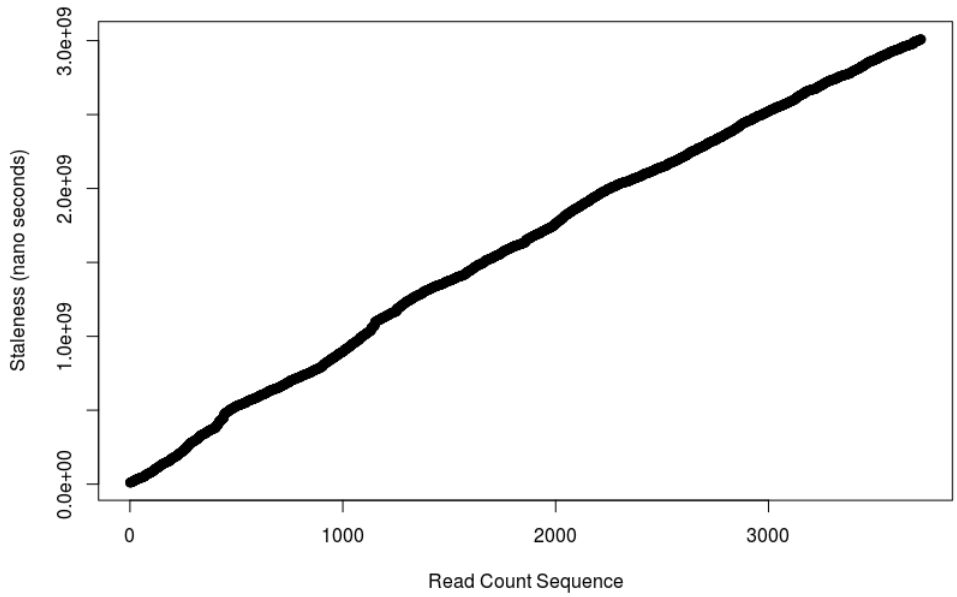


Figure 4.11: Staleness at 3 sec for Workload B. Single window between updates to read replica.

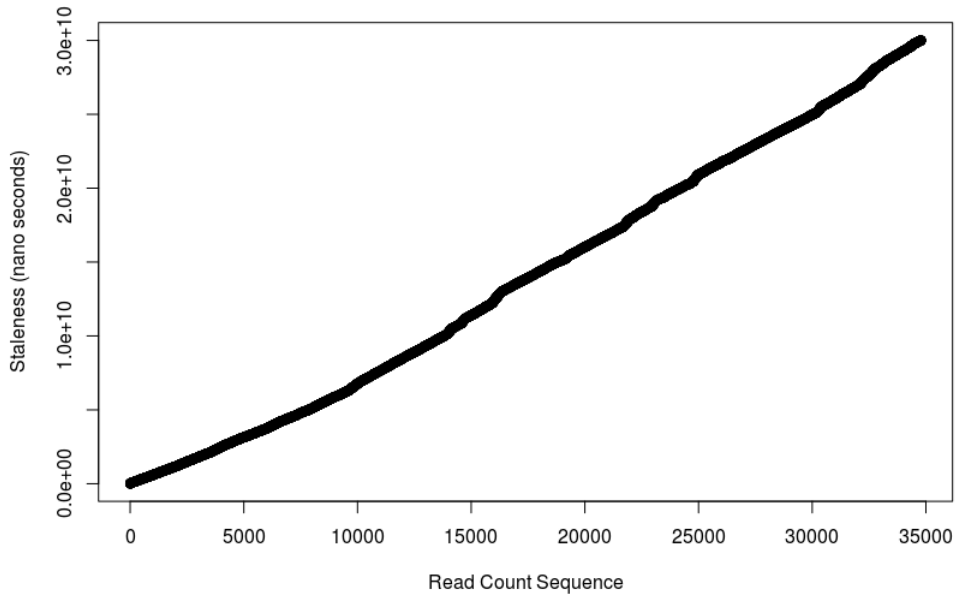


Figure 4.12: Staleness at 30 sec for Workload B. Single window between updates to read replica.

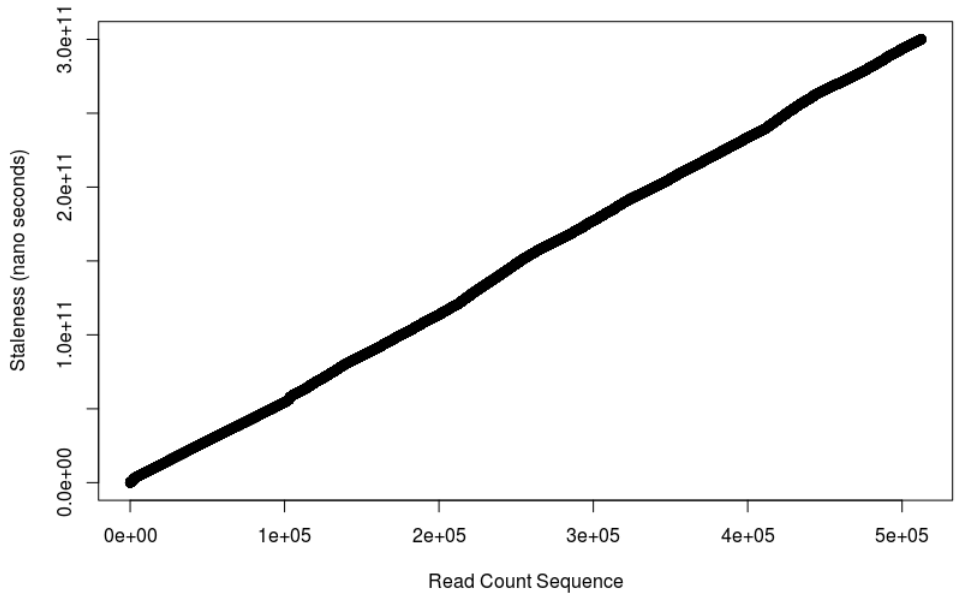


Figure 4.13: Staleness at 300 sec for Workload B. Single window between updates to read replica.

In Figure 4.14, Figure 4.15 and Figure 4.16 we display our evaluation of the absolute freshness for each read transaction. The results that we report are the aggregate counts of fresh/stale reads within one window between two refresh operations on the read replica. We do not report on the distribution of individual fresh/stale flags per read transaction, but we note that it follows a pattern similar to the reported for absolute timeliness in a later section (see Figure 4.26, Figure 4.27 and Figure 4.28).

Unlike other measures evaluated until this point, which depend solely on values that are calculated before the current window, the absolute freshness relies on tracking a list of updated values after t_u. In this sense the measure can be distinguished from others by not being an estimate but an accurate evaluation[1]. We also note, that by assigning a binary measure of freshness to individual items it provides a result similar to that of the absolute timeliness.

[1]However, the measure does not track the values of items themselves, just the fact that there was an update to an item. As a result, it can consider one item to be not fresh even if the recent updates have assigned the exact same value it possesses in the read replica.

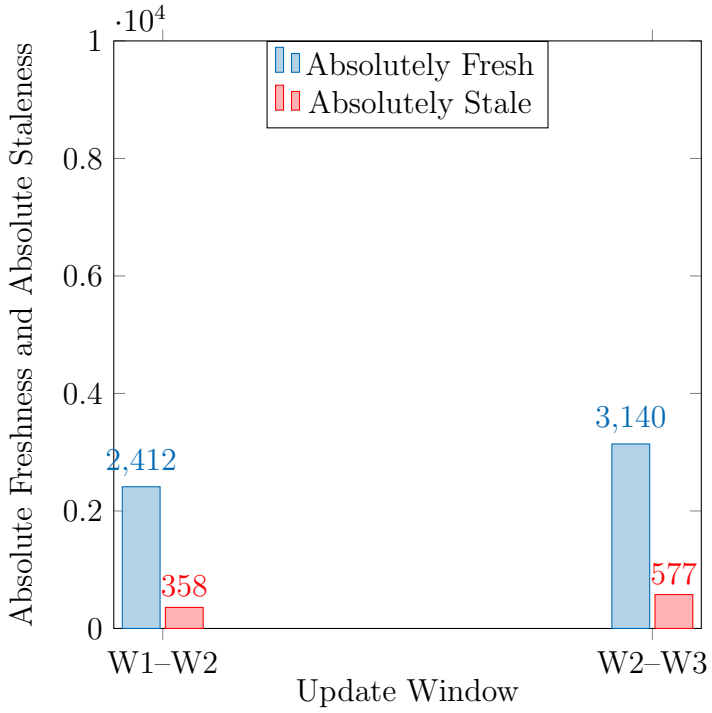


Figure 4.14: Absolute Freshness and Absolute Staleness at 3 sec for Workload B

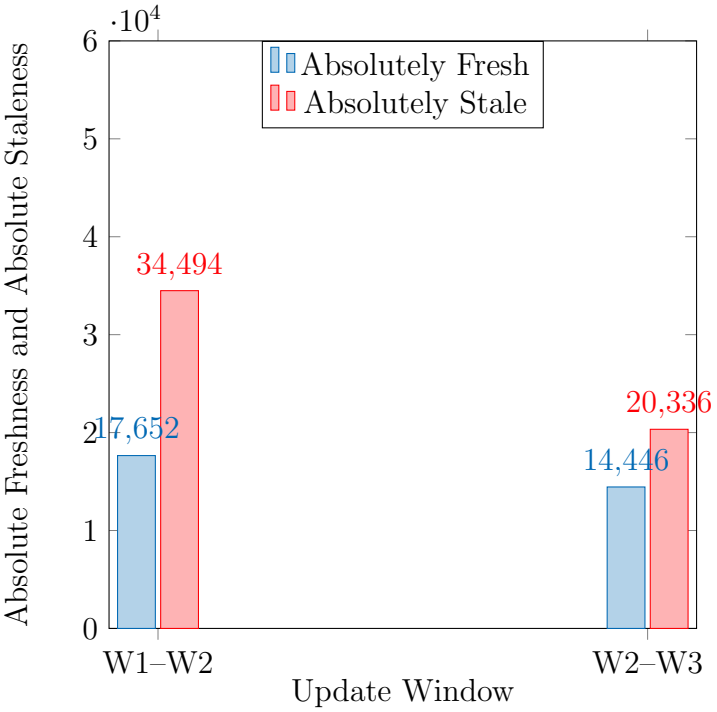


Figure 4.15: Absolute Freshness and Absolute Staleness at 30 sec for Workload B

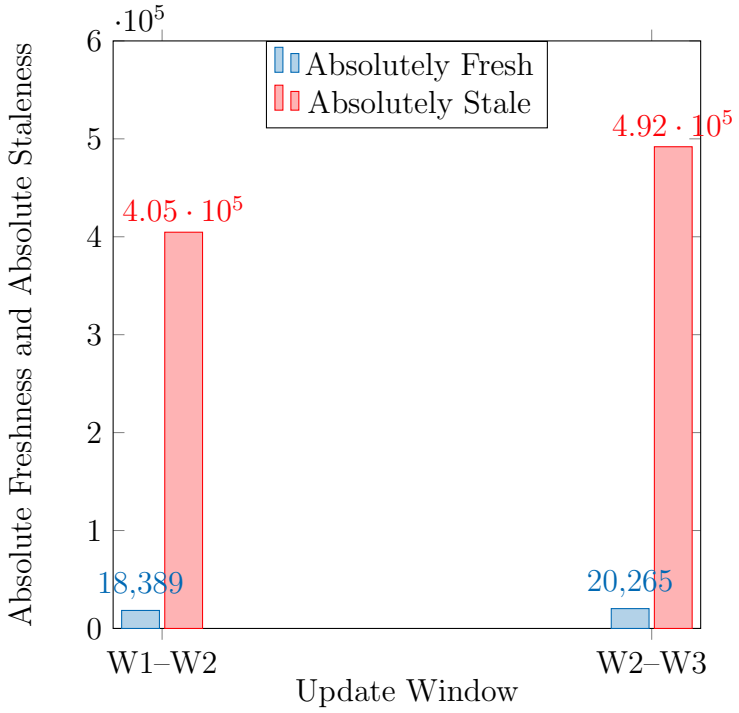


Figure 4.16: Absolute Freshness and Absolute Staleness at 300 sec for Workload B

4.3.3 Freshness Rate and Freshness Index

As defined earlier, the *freshness index* [RBSS02] is proposed as a measure that is more specific to time than to read transactions, and that reflects how much data deviates from the up-to-date version. If the freshness index is 1 then most read transactions have been on up-to-date data, else when the index is near 0 then most read transactions have been done on outdated data.

In Figure 4.17, Figure 4.18, Figure 4.19 we measure the freshness index across the read count sequence for the different refresh rates. Specifically we display the results of a single window of read requests between two batch updates to the read replica.

When comparing the figures the first noteworthy aspect is the difference in the X-axis. This indicates the amount of reads within the window of the two batch updates to the read replica.

We see that most cases match the overall throughput: in Figure 4.18, with a throughput of roughly 1000 transactions per second we can expect to find approximately 30000 operations within a window of 30 seconds and in Figure 4.19 approximately 300000 read transactions in 300 seconds. However, for the window selected in Figure 4.17, there is a lower behavior than expected (i.e. we would've expected to see 3000 reads in 3 seconds). This behavior occurs in some windows due to the overheads from the pessimistic concurrency control with locks, which are more evident in this case (in comparison to lower refresh rates), causing some windows to show a lower throughput

than what is seen in the complete workload. We discuss about this in a later section (Section 4.3.8).

Comparing the index measures itself, all three figures show a gradual decrease from 1 (i.e., 100% fresh) to close to 0.992 (i.e., 99.2% fresh). There is also a certain amount of scaling involved, as there is a consistent change of magnitude in the percentages as evaluated between the 30 and 300 seconds. On the other hand, in the measurements between 30 and 3 seconds, there is a larger difference in magnitude of 3 orders. This might be related to our observations regarding locking (i.e. since the throughput is reduced in the window, the freshness is kept specially high).

Also, concerning the index itself we note that the average freshness index is decreasing through the executions, which is consistent with the expectation that the correlation with the throughput would be negative. We discuss about this in a later section (Section 4.3.8).

It should be noted that all measurements that conform these apparently continuous figures, are individual observations, with gaps between them, as seen in Figure 4.17. The gaps between the observations correspond to interleaved updates (which increase the t_{lu}, leading in turn to step-wise decreases in the freshness index). When the measurement scope is larger, these gaps become invisible in figures.

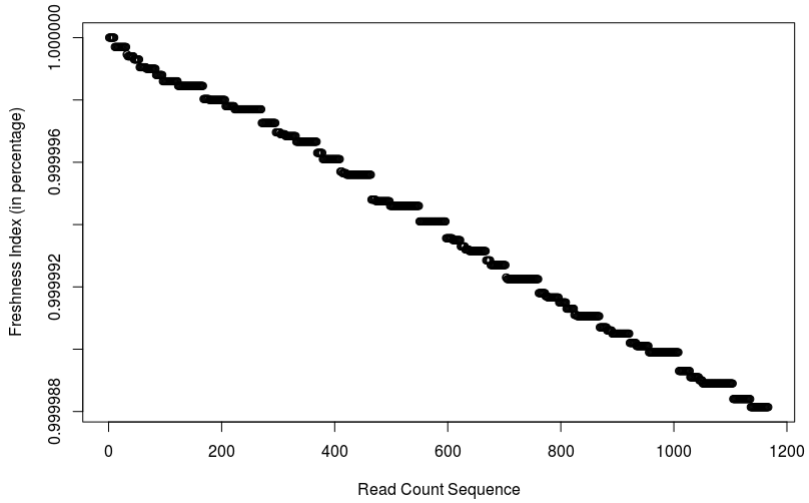


Figure 4.17: Freshness Index at 3 sec for Workload B. Single window between updates to read replica.

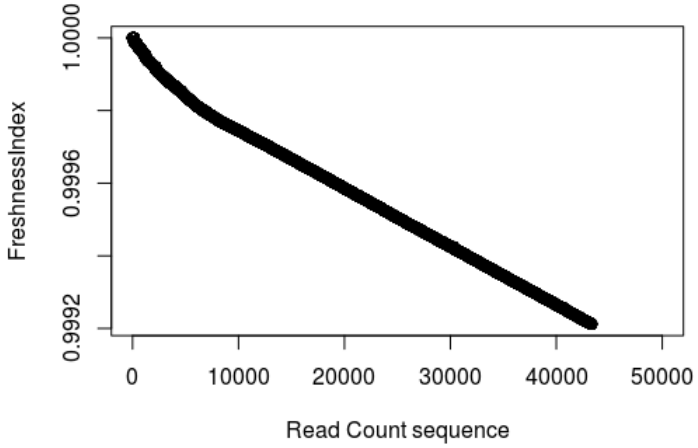


Figure 4.18: Freshness Index at 30 sec for Workload B. Single window between updates to read replica.

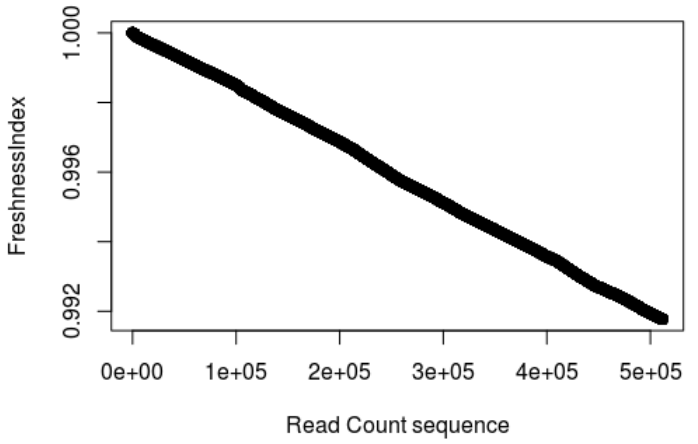


Figure 4.19: Freshness Index at 300 sec for Workload B. Single window between updates to read replica.

Moving on to the *freshness rate*, we report in Figure 4.20, Figure 4.21 and Figure 4.22 a trend that is similar in some aspects to the observed for the freshness index, but that is also markedly different in others.

In terms of what is similar between these metrics, we observe that the x-axis displays an equivalent number of read counts per window, and that the distribution of reads

per window corresponds to what is expected for the throughput in the cases of 30 and 300 seconds, but that it also diverges for the case of 3 seconds, due to the slight non-determinism in concurrency management.

Other similar observations pertain to: the granularity of the measurements, which are more visible in the 3 seconds case but less in the other cases and the overall downward trend of the freshness rate itself, which in turn leads to lower averages that match our expectation of a negative correlation with the throughput (to be discussed in a later section).

However the measurements for freshness rate are markedly different from those of the freshness index, since the first enable to report faster decays, going to 20 % and 0% in the cases of 30 and 300 seconds, respectively. This yields results that at 300 seconds, by the end of the time freshness rate falls to 0. With 30 seconds the freshness rate does fall but only to 20%. After every refresh the freshness increases to 100% and again behaves the same as before.

In fact, figures for the freshness index show a behavior that is roughly linear, while figures for the freshness rate move from close to linear to a shape that resembles exponential decay. Though both metrics introduce a non-linear component, in the form of non-constant divisions, in our study it is only the freshness rate which is able to model non-linear behavior.

This distinction means that the metric, by not being time dependent, but instead building upon the proportion of items changed w.r.t those in the read replica, is able to provide more insights than the freshness index for use cases which involve intensive updates, as the one we study. Thus we observe from our results that the freshness index, might be more applicable for domains where the number of updates is less important than the time difference. Considering that this is not the case for HTAP benchmarks, we suggest that freshness rate is a more appropriate measure.

Comparing the requirements from both measures, we note that both are quite similar in the number of values that they need to track. While the freshness index needs to store the maximum transaction time in a read replica (a fixed value per window), and the last update of each item in the write replica, freshness rate requires the number of items in a read replica (also a fixed value per window), and the number of items actually updated in a write replica.

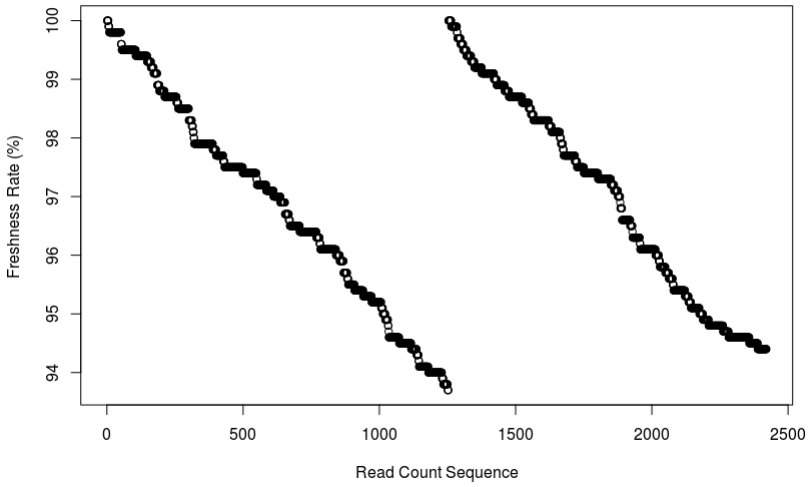


Figure 4.20: Freshness Rate at 3 sec for Workload B. Two windows between updates to read replica.

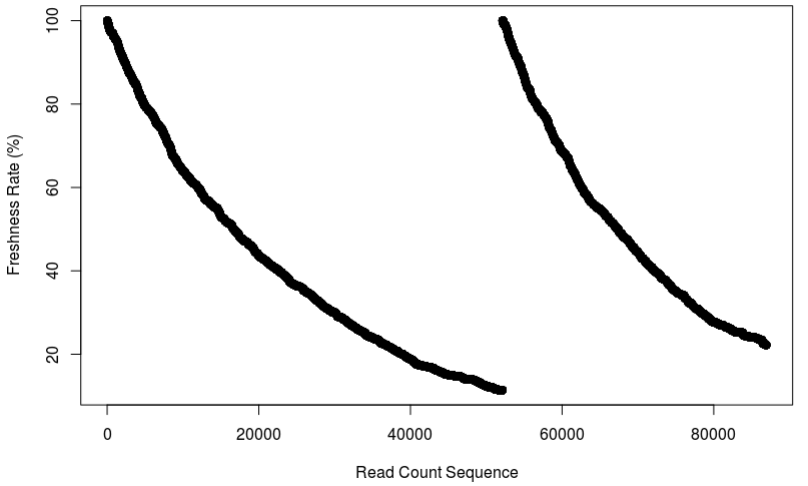


Figure 4.21: Freshness Rate at 30 sec for Workload B. Two windows between updates to read replica.

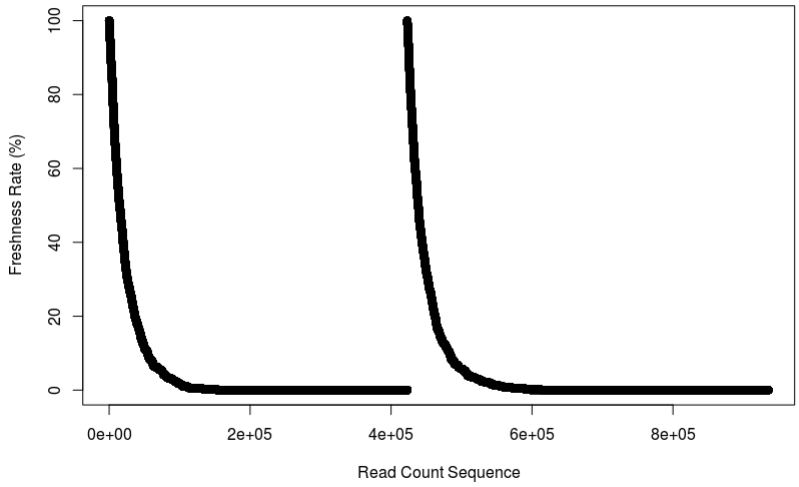


Figure 4.22: Freshness Rate at 300 sec for Workload B. Two windows between updates to read replica.

4.3.4 Obsolescence

As mentioned previously, *obsolescence* captures the age of the read replica with respect to the number of updates that have taken place in the write replica, since the last synchronization of the read replica. Our evaluation for this measure over two windows is presented in Figure 4.23, Figure 4.24 and Figure 4.25. The results confirm the expectations that obsolescence would follow the opposite trend of freshness rates and freshness indexes, increasing with time rather than decreasing. The trend is also linear and does not present exponential behavior (which stems from the linearity of the formula).

In terms of the different timestamps, we observe that the number of read operations per window is consistent with the reported throughput, with numbers being greater for the windows in the cases for 30 and 300 seconds. In addition, the number of writes in the results (i.e., the tx count on the y-axis) match well the percentage of writes expected for the workload. The number of writes in a 5% proportion to the workload would be, for 3000, 30000 and 300000 expected operations, roughly 150, 1500 and 15000, respectively.

When compared with other formulas, obsolescence is based on a linear function over the increasing number of transactions that arrive into the write replica. Gouging this metric is quite simple, as it only requires to track the time of the last update to the read replica, and the number of transactions elapsed since. Among its drawbacks by considering solely an aggregated view on the transactions (i.e., disregarding the specificity of items written), it is less informative than measures which consider items, like the freshness rate, and might give insufficient information about freshness when there is update skew within a window (i.e. when a small number of items receives a large number of updates).

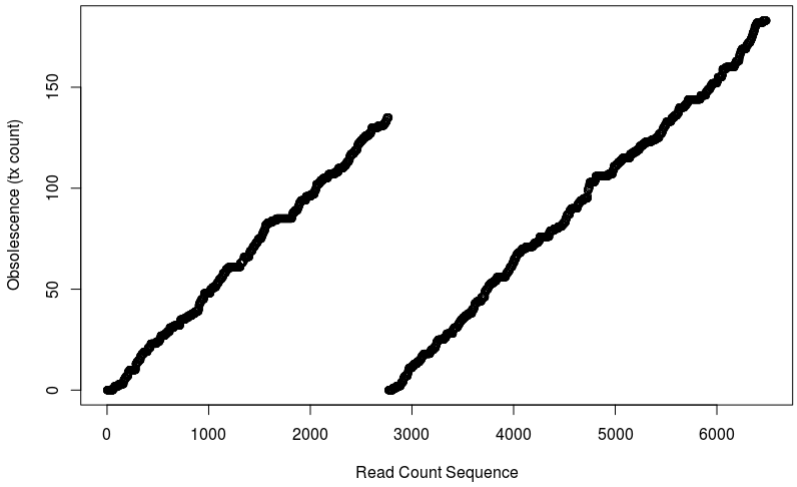


Figure 4.23: Obsolescence at 3 sec for Workload B. Two windows between updates to read replica.

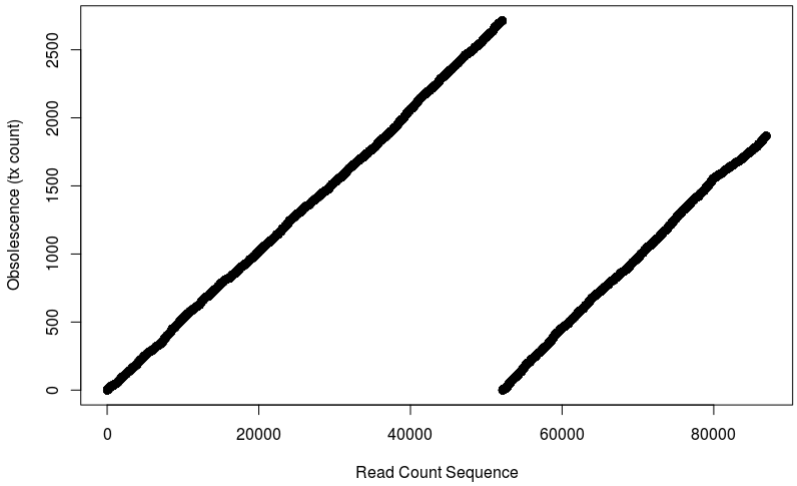


Figure 4.24: Obsolescence at 30 sec for Workload B. Two windows between updates to read replica.

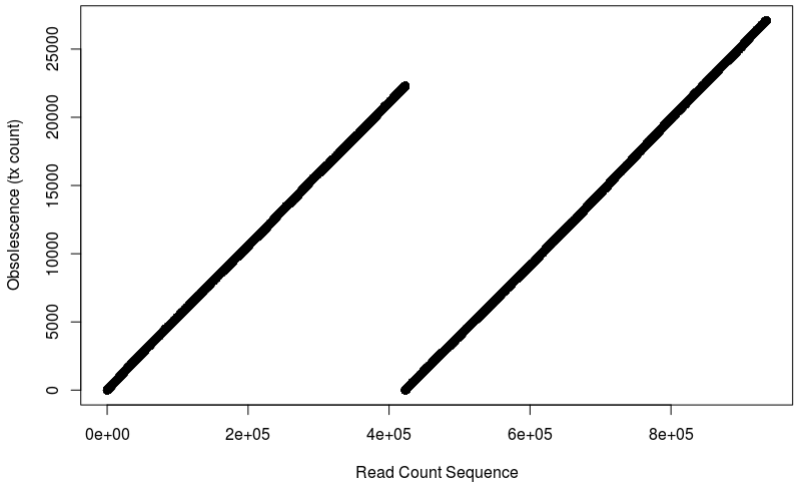


Figure 4.25: Obsolescence at 300 sec for Workload B. Two windows between updates to read replica.

4.3.5 Absolute Timeliness

Absolute timeliness is a measure that is applied to each read transaction, assigning to such transaction a score of 0 or 1, that indicates if the transaction is expected to be timely (i.e., no updates are likely since t_u) or not timely (i.e., updates might've occurred after t_u). To determine this score a heuristic predicts the volatility of items by using the time difference between their last two updates before t_u.

We present our study of this metric in Figure 4.26, Figure 4.27 and Figure 4.28. Specifically we show the scores assigned to each read transaction that arrives within a window, for refresh rates of 3, 30 and 300 seconds, respectively.

The first noteworthy aspect from these measurements is that we observe a densification of the values that are seen as untimely, starting from Figure 4.26, where they are relatively seldom, and increasing through Figure 4.27 and Figure 4.28, with the latter showing too the deterioration of the timely reads, which no longer manage to occur through a complete window. In this sense, by using this metric we achieve similar insights to those provided by the freshness rate: we see that there is a drastic loss of data quality taking place, due to staleness.

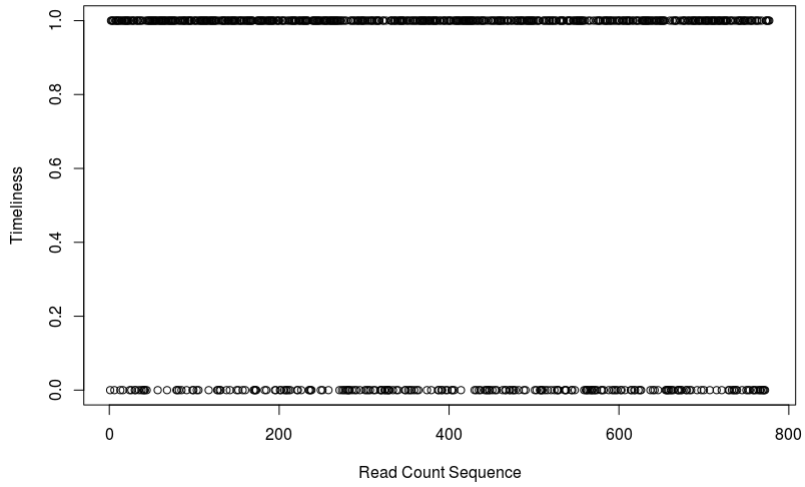


Figure 4.26: Absolute Timeliness at 3 sec for Workload B. Single window between updates to read replica.

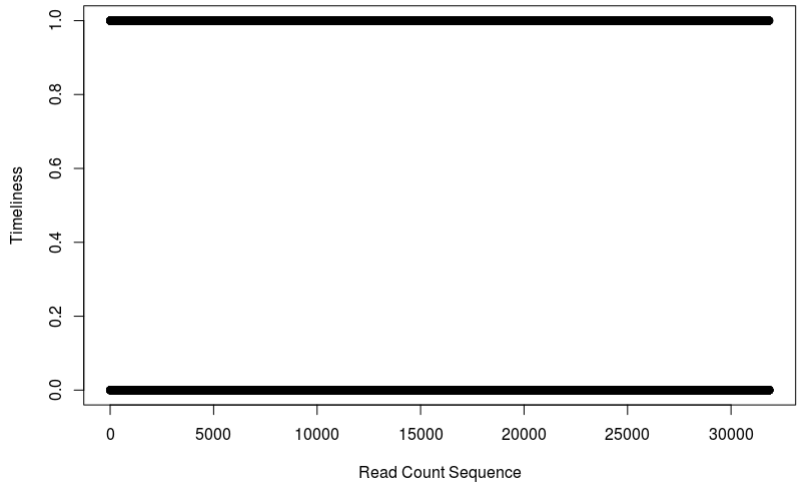


Figure 4.27: Absolute Timeliness at 30 sec for Workload B. Single window between updates to read replica.

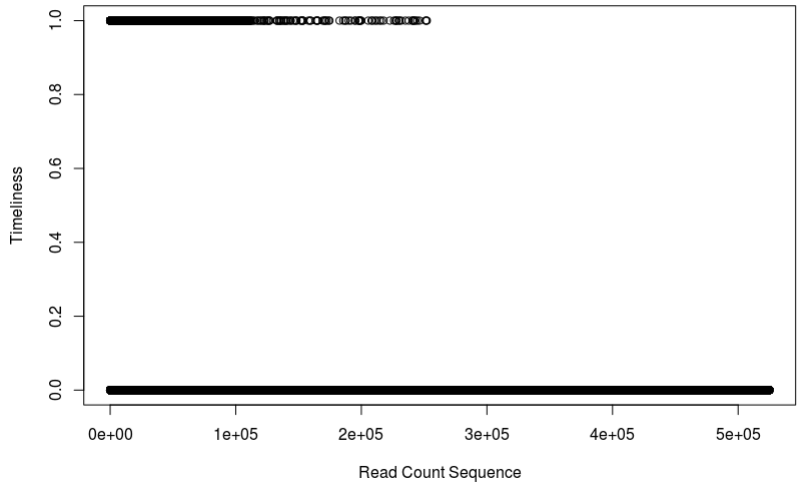


Figure 4.28: Absolute Timeliness at 300 sec for Workload B. Single window between updates to read replica.

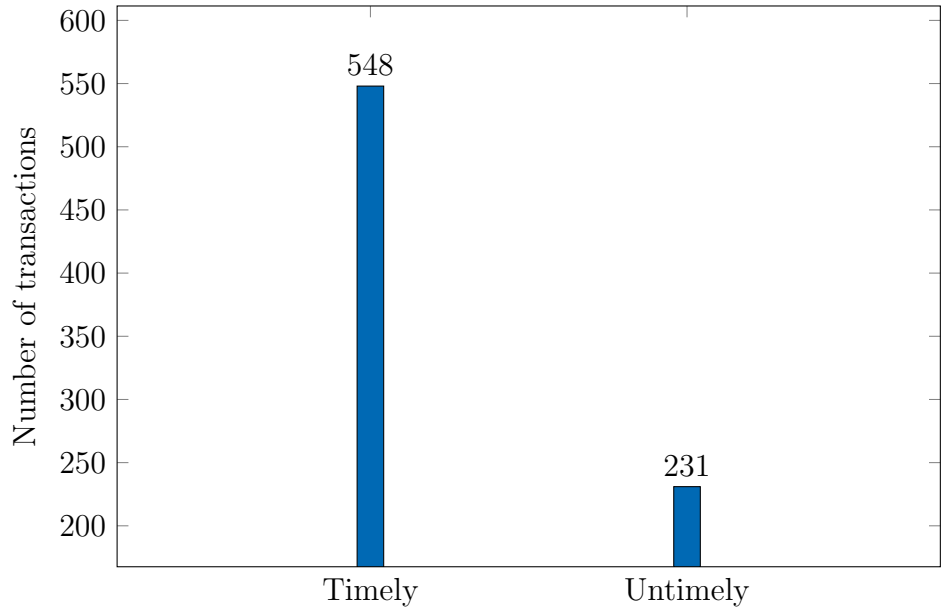


Figure 4.29: Absolute Timeliness at 3 sec for Workload B (Totals). Single window between updates to read replica.

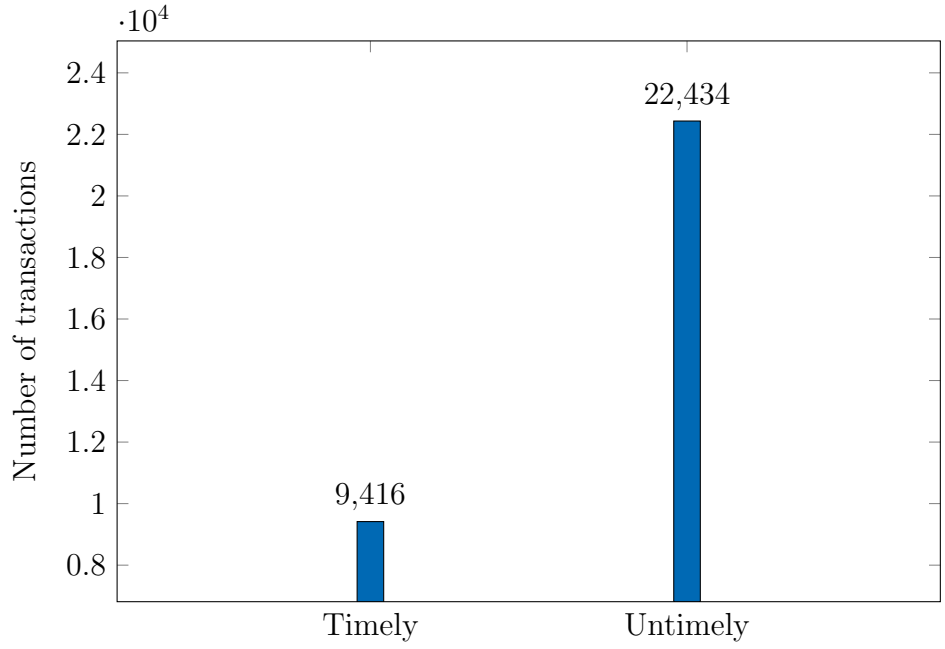


Figure 4.30: Absolute Timeliness at 30 sec for Workload B (Totals). Single window between updates to read replica.

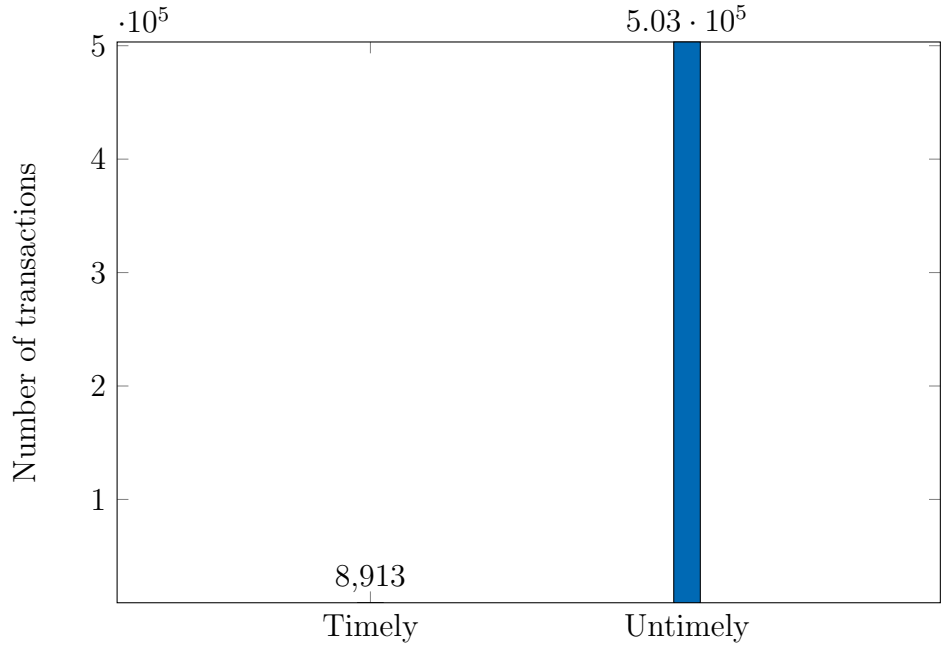


Figure 4.31: Absolute Timeliness at 3 sec for Workload B (Totals). Single window between updates to read replica.

The precise numbers for how many items fall, by the end of the window, in each category (i.e., timely or not) are displayed in Figure 4.29, Figure 4.30 and Figure 4.31. Timely reads constitute, respectively, 70%, 30% and 0.02% of the observed cases.

Regarding the number of operations per window, the results are consistent with our expectations.

To conclude our discussion of this metric we should note that it requires to track t_u, in addition to the volatility of items. This last factor can be calculated with the last two updates (which requires more information to be kept), which could also be pre-computed. Perhaps better results can be achieved by considering more information from the item's history of updates to calculate the volatility. We should also note that the volatility of an item in the past window might not reflect precisely how it is updated in the current window.

4.3.6 Three more measures for Staleness, based on the items update history

In Figure 4.32, Figure 4.33 and Figure 4.34 we present our results for the Enhanced Averaging method of Staleness. We specifically plot the staleness estimated for each read transaction on an item.

Our first observation pertains to the ranges of the values. We note that for some read transactions, the values predicted are much larger than those of the *Staleness*. We expect this to be explained for items in which the expected update was later than t_{lu}, but smaller than t_{maxtx}.

Across the refresh rates we also observe a densification process that tends to segregate specially cold items to the bottom of the plot, specially hot items to the top, and the majority in the middle. The process seems similar to *Currency (alternative)*, with the distinction that the *Enhanced Averaging Method* seems to predict lower stalenesses for most of items.

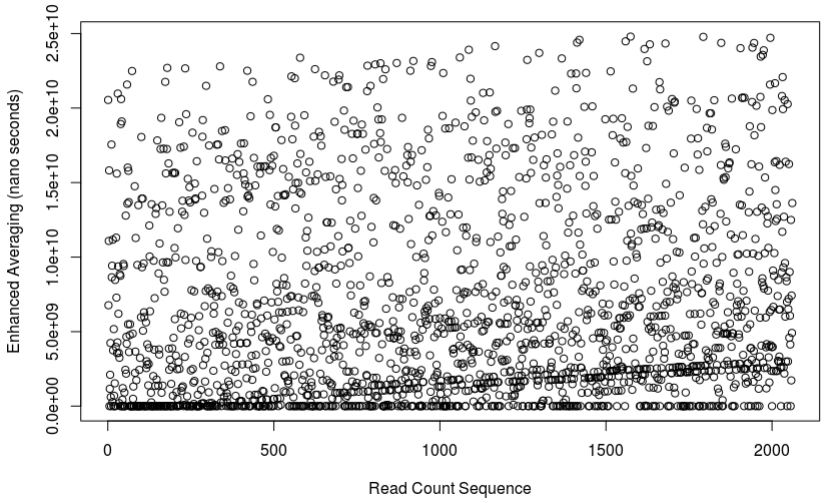


Figure 4.32: Staleness with Enhanced Averaging Method at 3 sec for Workload B. Single windows between updates to read replica.

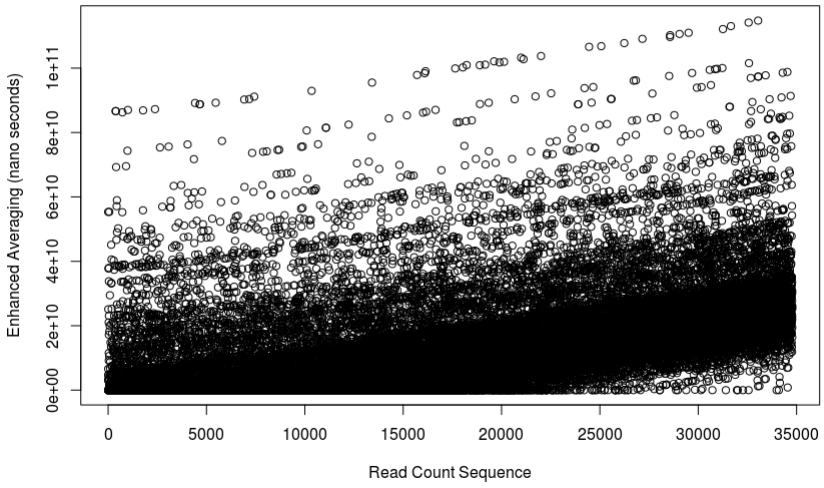


Figure 4.33: Staleness with Enhanced Averaging Method at 30 sec for Workload B. Single windows between updates to read replica.

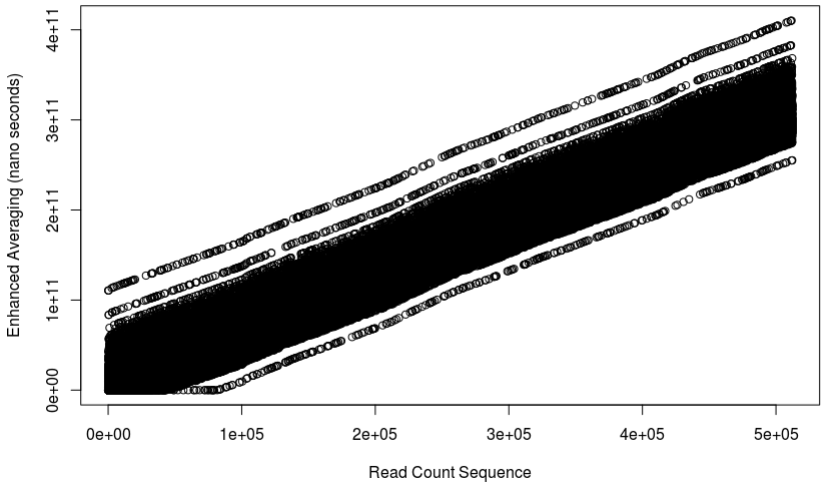


Figure 4.34: Staleness with Enhanced Averaging Method at 300 sec for Workload B. Single windows between updates to read replica.

In Figure 4.38, Figure 4.39 and Figure 4.40 we show our results for the shifting window method. We display our results for a window of 0.75 seconds, which is smaller than the refresh rate. If we would've selected a window larger or equal to the refresh rate, the results would have been close to those of the enhanced averaging (this is, if we interpret the formula to only use results from the previous window[2]).

Our evaluation for the shifting window does not show a meaningful sense of staleness. When choosing a window size lower than the refresh rate the formula mixes the update rate of the previous window with the expectation that t_{lu} happened there. Hence we find no workaround to adopt such formulation to our use case.

[2]When we interpret the formula to use also information from the current window, the formulas target application becomes unclear.

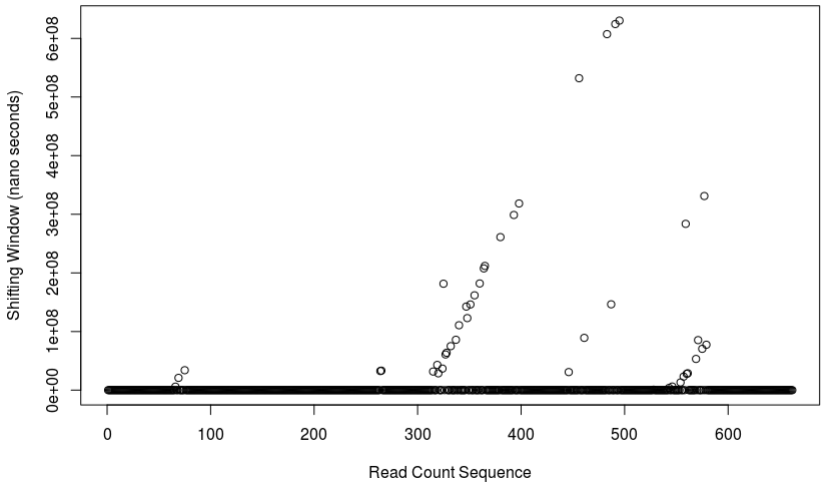


Figure 4.35: Shifting Window Method at 3 sec for Workload B. Single windows between updates to read replica.

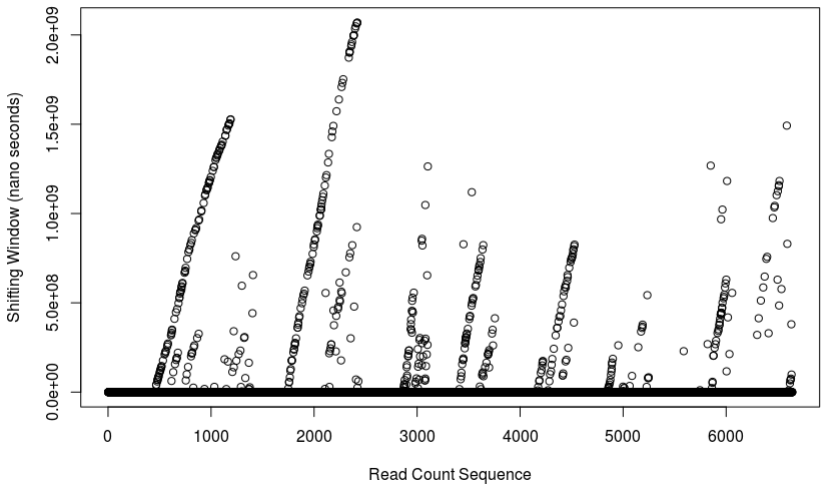


Figure 4.36: Shifting Window Method at 30 sec for Workload B. Single windows between updates to read replica.

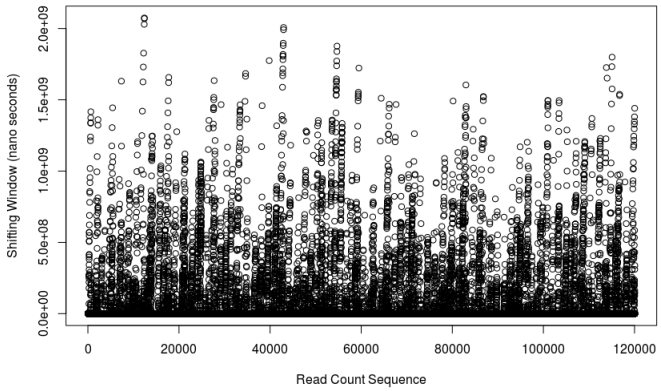


Figure 4.37: Shifting Window Method at 300 sec for Workload B. Single windows between updates to read replica.

We include, for illustration, in Figure 4.38, Figure 4.39 and Figure 4.40 the results for the Sav_{ti} for a selected item on different windows using the $Staleness_{ExponentialSmoothing}$. We use the same window size than for the former approach. Our results, once again do not show any meaningful information. This leads argument to the strong observation that the last two measures are either not applicable, nor relevant for our use case. The first one since it can presuppose a mix between updated data and predictions which is not appropriate for our use case and damages the results, the second simply loses relevance because Absolute Timeliness provides accurate measures, and in an evaluation with a window smaller than the refresh rates there are no clear results.

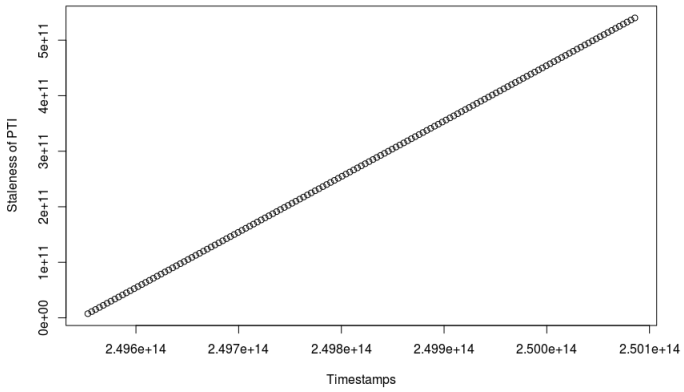


Figure 4.38: Exponential Smoothing Method at 3 sec for Workload B. Single windows between updates to read replica.

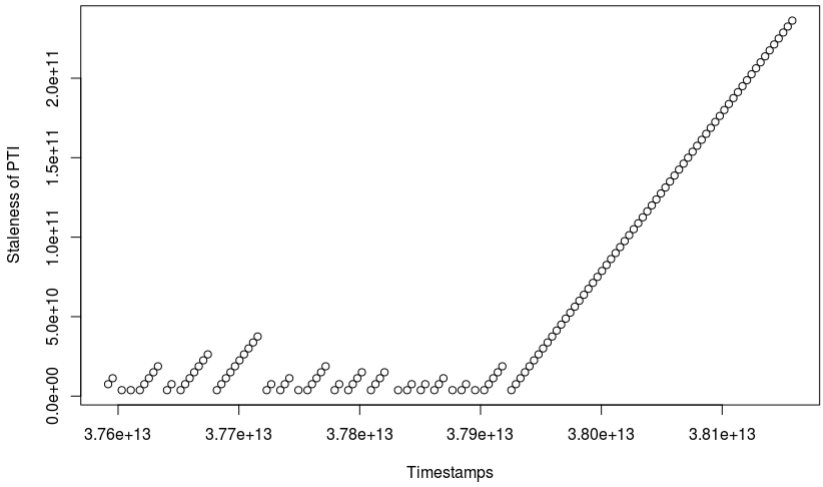


Figure 4.39: Exponential Smoothing Method at 30 sec for Workload B. Single windows between updates to read replica.

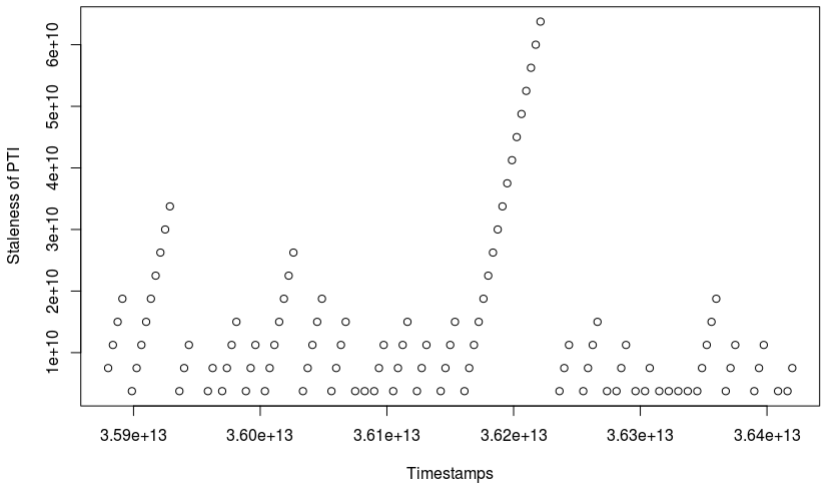


Figure 4.40: Exponential Smoothing Method at 300 sec for Workload B. Single windows between updates to read replica.

With the results from these measures we conclude our evaluation. In the next section we discuss how the measures fared in their correlation to the throughput.

4.3.7 Correlation of freshness measures with throughput

In Table 4.3 we present how these measures correlated with the throughput in our study[3]. Due to their limitations, we do not include results for Shifting Window and Exponential Smoothing.

For almost all cases we selected the average of the measures through the window. For Absolute Freshness and Absolute Timeliness we report the results for the final proportion of fresh and timely reads within the windows, respectively.

[3]Please note that in Table 4.3 we include *Timeliness Rate*. This measure is presented and discussed in Section 4.3.8.

Measure	**3 sec**	**30 sec**	**300 sec**	**Correlation**
Throughput (ops/s)	1040	1080	1190	
Currency	1624615483	14565821335	162245907132	0.9824095804
Currency (alternative)	2690518185	21659208317	160485842034	0.9887554057
Staleness	1607485517	13912662775	149585455088	0.9828588602
Absolute Freshness (Proportion of Fresh Reads)	0.855865577	0.3692481133	0.04133078851	-0.9304351059
Freshness Index	1	1	0.99999994	-0.9662823901
Freshness Rate	92.86876	44.460705	5.096996	-0.9484133876
Obsolescence	80	1186	12418	0.983964925
Absolute Timeliness (Proportion of Timely Reads)	0.703465982028241	0.29563579277865	0.0174014057008981	-0.931670848
$Staleness_{EnhancedAveraging}$	6651911865	14144457135	149871482125	0.9772044222
Timeliness Rate	99.09637	43.932625	2.243475	-0.9416382553

Table 4.3: Correlation of measures to the observed throughput

For all measures we report correlations that are close to 1 or -1. This finding corresponds well to the studied correlation between freshness and throughput in systems like the one we study.

All metrics correlated in the direction expected.

4.3.8 Discussion

In this section we discuss our observations regarding these measurements. We start by providing general observations regarding where the measurements could be applied. Next we note the insufficiency of global measurements and of time-based measurements that do not consider item volatility. We conclude recommending measures and proposing an extension to them.

4.3.8.1 Measures can be classified according to their use of information posterior to t_u

Our first observation, stemming from the practical experience of implementing the measures, is that we see measures can be divided into two groups, based on their use of information posterior to t_u or not. Among the measures that use it we include: *Absolute Freshness, Freshness Index, Freshness Rate and Obsolescence*. The rest of measures (save for $Staleness_{ShiftingWindow}$ and $Staleness_{ExponentialSmoothing}$) fall, logically, in the other category.

$Staleness_{ShiftingWindow}$ assumes a mix of a posteriori information with an approximation of apriori information. We find that this is a use case which has no clear application.

The distinction between measures into these two large groups is important because it establishes the way in which these measurements can be applied and the strategies that can be adopted to model per-item freshness.

Measurements that use information posterior to t_u are preferred for adoption in benchmarks. They can be computed after the execution of the tests, by analyzing the logs.

Furthermore the use of this a posteriori information can help the measures provide precise insights (i.e., not based on estimations), such as the case of the *Absolute Freshness* and *Freshness Rate*, which count as fresh exactly those items that have been updated after t_u.

Measures that do not use information posterior to t_u are suitable to live systems, and are to be preferred for HTAP designs that might employ freshness measurements as part of the online statistics used by query optimizers.

Paired with their usability, these measurements are tasked with the challenge of properly modeling the expected updates of items in the current window, by relying only on past information of the items. We have seen some ways in which this is modeled within the studied measures, such as the *Currency (alternative)*, where the maximum transaction

for a given item before t_u is used to model the trajectory for the currency of items. We have also seen this in *Absolute Timeliness*, where the two last updates to an item before t_u are employed to define the volatility of the item, and thus determine if an item is fresh or not at the moment of a read transaction.

4.3.8.2 Global measures and time-dependent measures which disregard the volatility of items provide insufficient information for our use case

We have also seen that some measures are global (or item-independent), such as *Currency, Freshness, Staleness and Obsolescence*. From our study we note that these measures provide insufficient information for our use case, when compared to item-based ones: they do not give insights into the state of the read replica as a whole, nor into the exact freshness of a read transaction. Thus we strongly encourage the avoidance of these measures, since better measurements could be employed at similar costs.

The same statement can be applied to formulas that are based on time observations while disregarding the volatility of items (i.e. such as *Currency, Freshness, Staleness, Freshness Index*); thus they are not adequate for a write-intensive scenario when the number of updates is more informative than the time elapsed. In our evaluation we adopted a workload with few updates (5%), the limitations of time-based measures should be even more marked when the updates are more.

4.3.8.3 Among the metrics that use information posterior to t_u, *Absolute Freshness* and *Freshness Rate* should be preferred

Among the metrics that count with information *a posteriori* to t_u we consider *Absolute Freshness* and *Freshness Rate* to be the most useful. In what follows we detail our case.

Obsolescence, though easy to track, lacks the granularity of items, thus it is not precise. The *Freshness Index*, though predicated as a non-linear formula, which should be able to model decay, has the limitation that by being restricted to time measurements is hardly applicable to write-intensive scenarios (when the number of operations reports better on the amount of change than does time). For example, in our study, we observe that across freshness changes (see Figure 4.17, Figure 4.18 and Figure 4.19) the metric does not report a marked decay. This limitation is notable when compared to the *Freshness Rate*, which truthfully reports a more drastic deterioration in our evaluation (see Figure 4.20, Figure 4.21 and Figure 4.22).

Absolute Freshness measures with complete certainty the quality of the read operations in a workload past. It focuses on the quality of the operations rather than on that of the items. The *Freshness Rate* complements well the *Absolute Freshness* because it tracks updates to items and reports on the freshness (i.e., quality) of the read replica as a whole. As a result we recommend the selection of both these measurements, since each provides a different insight into the storage and operation quality.

4.3.8.4 Among the metrics that do not use information posterior to t_u, *Absolute Timeliness* should be preferred

Considering the metrics that use solely information *a priori* to t_u, we strongly encourage the adoption of measurements that model the volatility of items, like *Absolute Timeliness*.

Currency and *Freshness* show the limitations of global measures, not considering items nor volatility, and in addition, with the latter being almost entirely uninformative on its own. *Currency*, to an extent, proposes simply a change of time coordinates for the study of freshness. However, as noted, the time elapsed is not as informative as the number of updates.

Currency (alternative) improves over *Currency*, but it does not provide a mechanism to determine if a read is fresh or not, nor can it be employed straight-forwardly to determine the quality of the read replica. The same problem applies to $Staleness_{EnhancedAveraging}$ and $Staleness_{ExponentialSmoothing}$.

Among these measurements, only *Currency (alternative)*, $Staleness_{ExponentialSmoothing}$ and *Absolute Timeliness* attempt to model the volatility of items. The former employs a simple model, based on the last update of each item before t_u, the latter estimates it better by considering the time elapsed between the last two updates to an item before t_u. As a result the latter presents a better model for how likely is an item to be fresh at the time of a read transaction. In consideration of this, and the fact that *Absolute Timeliness* gives an evaluation on the quality of a read transaction, we recommend this as a measure to employ from the group of those that rely on information *a priori* to t_u. $Staleness_{ExponentialSmoothing}$ provides a sophisticated model of volatility, however since it does not provide a clear measure like Absolute Timeliness, we do not consider to be specially relevant.

4.3.8.5 *Timeliness Rate*: our proposal to complement *Absolute Timeliness*, leveraging the volatility model

Stemming from the observation that *Absolute Timeliness* does not convey information on the state of the read replica, we propose the use of another companion measure. This is a novel measure which we call *Timeliness Rate*, and can be calculated with the following formula:

Formula: Timeliness Rate$(t_i) = nmv_{nnt}/nmv$.

Where nmv_{nnt} is calculated as: $nmv - n_{nt}$, with n_t: number of items marked as absolutely untimely.

Figure 4.41, Figure 4.42 and Figure 4.43 display the results for evaluating the *Timeliness Rate* in our experiment. Using the same windows than for *Freshness Rate*, we observe that our measure is able to convey a similar information, though the model predicts, at the end of the window less decay in freshness than the real one (i.e. 99% vs 94%, 32% vs 17%, 0 % vs 0 %). For comparison, see Figure 4.20, Figure 4.21 and Figure 4.22.

The same trends are preserved in all plots, except for the freshness rate of 3, where the decline modeled is notably less steep than the real one.

The absolute errors w.r.t the means of the *Freshness Rate* are: 6.22761, 0.52808 and 2.853521, for the refresh rates of 3, 30 and 300, respectively[4]. In addition, the negative correlation with throughout is preserved and the difference when compared to the correlation of the *Freshness Rate* is 0.006775132314.

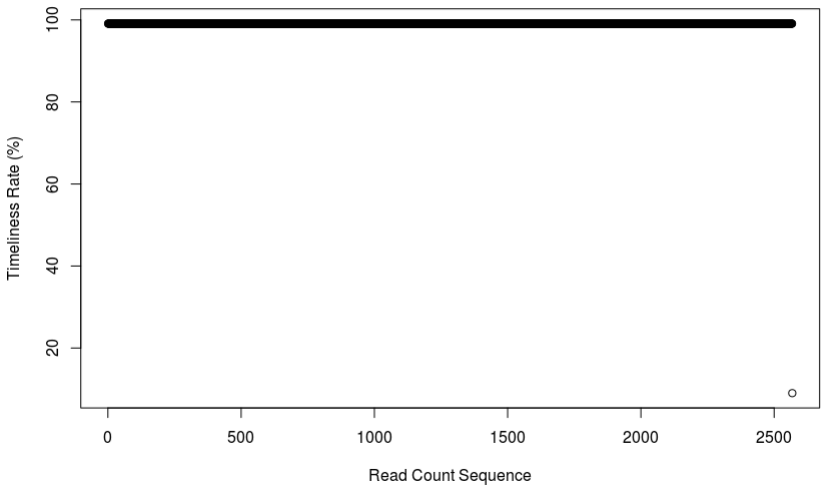


Figure 4.41: Timeliness Rate at 3 sec for Workload B. Single windows between updates to read replica.

[4]For reference, the values are presented in Table 4.3.

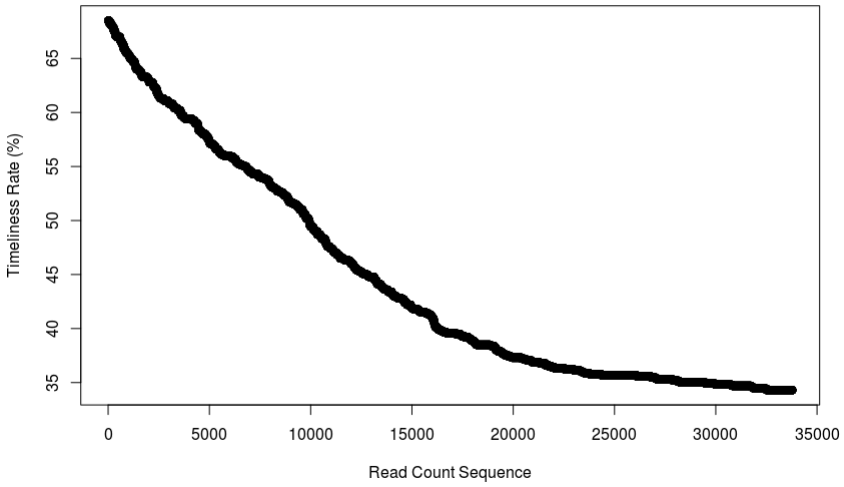


Figure 4.42: Timeliness Rate at 30 sec for Workload B. Single windows between updates to read replica.

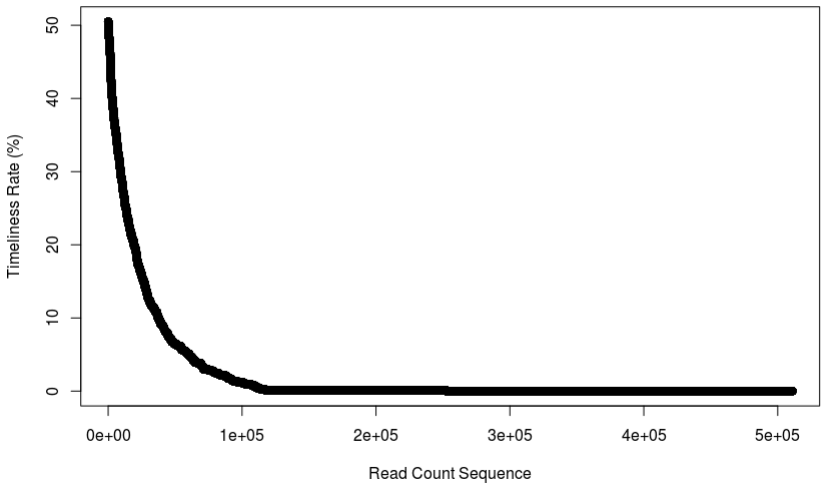


Figure 4.43: Timeliness Rate at 300 sec for Workload B. Single windows between updates to read replica.

These findings validate that the volatility model used in the calculation of *Absolute Timeliness* is appropriate for modeling the freshness of HTAP databases, leading to a close approximation of the real values when using this model to estimate the *Freshness Rate*.

A complete comparison of *Absolute Timeliness* with *Absolute Freshness*, though pertinent, was left for future work.

4.3.8.6 Shifting Window Staleness is not pertinent to our use case, Exponential Smoothing Staleness, with its difficulties for aggregation falls short when compared to Absolute Timeliness

In our study of $Staleness_{ShiftingWindow}$ we found that a possible application is to employ it to improve over Absolute Timeliness. For this we evaluated on cases where the window size was smaller than the refresh rate. Such assumption lead to an inadequate expression of the measures, producing inappropriate results. On the other hand we found no clear way of organizing the measure such that it would work on a priori information only. Hence, the sophisticated method of $Staleness_{ShiftingWindow}$ is not pertinent to our use case, and instead Absolute Timeliness is to be preferred.

4.3.8.7 Some measures can be considered as stand-alone measures, while others can be considered building blocks

We conclude our discussion by noting that some measurements, by themselves, are able to answer (with varying degrees of goodness) the information need of evaluating the quality of read transactions (e.g. *Absolute Freshness, Absolute Timeliness*) and the quality of the read replica (e.g. *Freshness Rate*). Other measures serve only as building blocks to fulfill these and other information needs (e.g. *Currency, Currency (alternative), Freshness, Staleness, Freshness Index, Obsolescence*). As a result in our recommendation of metrics we do not discourage the use of others if they care helpful to fulfill other information needs apart from those considered for our study.

4.4 Summary

In this chapter we collected our evaluation of different freshness measures over an execution of the YCSB benchmark workload B (consisting of 95% reads and 5% updates) over an HTAP prototype called Blinktopus. We defined the general evaluation question which motivated our research, we presented the freshness metrics we found from the literature, discussing their computational requirements and summarizing their features. Next we evaluated the measures using the aforementioned configuration and displayed results over windows between two refresh operations to the read replica.

We noted the pros and cons of the measures in describing the quality of individual read operations and of the read replica as a whole, for measures that require information posterior to t_u and for measures that are tailored to not require such information. From the first group we were able to recommend *Absolute Freshness* and *Freshness Rate* as the preferred measures; from the second group we identified *Absolute Timeliness* as the best candidate. In addition, to supplement the need for a measure, we proposed and tested a novel measure, called *Timeliness Rate* so as to model the *Freshness Rate* when there is no information posterior to t_u, and a model for the volatility of items is available.

In the next chapters we study an existing HTAP benchmark and conclude our work.

5. Results from HTAP Benchmarks and Alternatives for Adding Freshness

In this chapter we present the results of our evaluation of the HTAPBench benchmark results against an OLTP system. We report results that do not replicate the expectations. Our results are merely an illustration, to complement our work and portray what can be accomplished by such a tool. These results do not add to our core evaluation question, save for suggesting how the freshness metrics might be integrated into a benchmarking tool.

- **Mixed workload:** In this chapter we give a brief introduction and then focus on the available mixed workload benchmark, HTAPBench and the results report provided by it Section 5.2.
- **Evaluation:** The experimental setup for OLAP, OLTP and Hybrid systems are discussed.
- **Summary:** We summarize the state of art of HTAPBench and suggest how freshness measures we recommend can be added to its reporting.

5.1 Introduction

The benchmarking approaches defined were especially to suit the existing traditional taxonomy of OLTP and OLAP systems. As discussed in Chapter-2, the benchmarking approaches TPC-C and TPC-E are tailored for OLTP workloads and the benchmarks TPC-H and TPC-DS are for analytical workloads (OLAP). As each of these benchmarks focuses on optimization challenges that are associated to each system type which defines

evaluation suites with different and contradicting motives. When an OLTP target operation is optimized, the OLAP performance would intensively degrades and vice-versa. Considering the scenario wherein specific set of queries that need distinct storage layouts to gain efficiency are generated by OLTP and OLAP workloads. Also, different sized datasets can be accommodated by taking the concept of warehouse as scaling factor.The optimization of storage layout must be attained in order to obtain efficient hybrid systems. Additionally the storage metrics generated by OLAP are sequential while the OLTP workloads are mostly random. Few other reasons along with the above mentioned ones cause the workloads to have evaluated independently. But this was changed by [CFG+11], where in an approach for HTAPBench providing a unified metric for HTAP systems geared towards execution of constantly increasing OLAP requests limited by an admissible impact on OLTP performance was proposed. This is accomplished by regulating the coexistence of OLTP and OLAP workloads using load balancer and further proposing a method such that new data and results are generated, so that across the runs the OLAP requests over freshly modified data are comparable.

A new metric was introduced by HTAPBench to provide the reading of analytical capability as system scales. A hybrid workload which exercises a workload with operational and analytical activity at the same time and over the same system was introduced. A client balancer was also introduced which controls the launch of analytical clients alongside ensuring that the OLTP activity stays within configured threshold and the results are kept comparable across runs by addressing data uniformity for workload. The user is allowed to configure the necessary workloads using the configuration file.

We have discussed different benchmarks in previous chapter, which are focused on individual workloads either on OLAP or OLTP workloads. The goal of these benchmarks is to evaluate the systems according to their optimization challenge. As each of these systems has different storage procedure and different access patterns, these systems are efficient in the way of achieving high throughput and high latency. Hence for these systems, OLTP and OLAP workloads are evaluated individually. In general, it is not easy to evaluate the mixed workloads in a single machine because in a HTAP system the throughput of one engine should not affect the others latency performance.

Authors Coelho et. all [CPV+17] proposed HTAPBench benchmark for the mixed workloads. This benchmark is a open source benchmark and to run this HTAPBench the database should be installed and it needs to be run on a server. To test this benchmark, we have used the *PostgreSQL*. The PostgreSQL is an OLTP database and it it is connected through a JDBC connection. The main goal of the HTAPbench is to define a unified metrics (Section 2.2.4.2) for the HTAP workloads. The unified metrics is nothing but maintaining the transaction motion on the increasing demand of analytical workers and to analyze the analytical queries. It measures the performance of OLTP and OLAP workloads with execution methods.Each query is executed five times. The SUT also manages the number of OLAP workers those are added to it without affecting the throughput of the OLTP.

The concept of configuration file has been used in HTAPbench [CPV+17]. It has been shown in Figure 5.1 and Figure 5.2 below. In this file, the benchmark is given some privileges to configure the HTAPbench. With the help of this benchmark, we can test the different databases and we can evaluate the system. For the evaluation and testing, we have considered PostgreSql database. In fact, different databases can be added for the evaluation according to the preference. The postgres created a htap database and created a connection between the HTAPbench and htapb database. Further, the data is loaded into the schema.

```
<?xml version="1.0"?>
<parameters>

    <!-- Connection details -->
    <dbtype>postgres</dbtype>
    <DBName>HTAPB</DBName>
    <driver>org.postgresql.Driver</driver>
    <DBUrl>jdbc:postgresql://localhost:5433/htapb</DBUrl>
    <username>postgres</username>
    <password>admin</password>
    <isolation>TRANSACTION_READ_COMMITTED</isolation>
```

Figure 5.1: HTAPBench configuration file [CPV+17]

```
<isolation>TRANSACTION_READ_COMMITTED</isolation>

<!-- HTAPB only-->
<target_TPS>1</target_TPS>
<error_margin>0.2</error_margin>

<!-- For a solo OLTP or OLAP workload-->
<!-- OLTP & OLAP only -->
<warehouses>35</warehouses>

<!-- For a solo OLAP workload-->
<!-- OLAP workers -->
<OLAP_workers>10</OLAP_workers>

<!-- The workload -->
<works>
    <work>
      <!-- time in minutes-->
      <time>120</time>
  <!-- First txn should always be the NewOrder - This is imposed by HTAPB -->
  <!-- The first 5 are the TPC-C transactions. The Remainder are the TPC-H queri
      <weights>45,44,4,4,3,3,2,3,2,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5</weights>
    </work>
</works>
```

Figure 5.2: HTAPBench configuration file [CPV+17]

Cole et al. [CFG+11] primarily proposed a unified metrics which enables the quantitative comparison of very similar systems, and also very different systems. Additionally they also proposed solutions to the important challenges of hybrid benchmarks such as the Client Balancer along with comparable and homogeneous results across executions. The proposed prototype has been tested for validation over the OLTP, OLAP and Hybrid systems. It has been shown that HTAPBench is capable of distinguishing different classes of systems along with distinguishing systems within same classes with high precision. This method concludes that it is able to introduce the required workload randomness

while keeping the results comparable by ensuring equal query execution costs across the whole dataset.

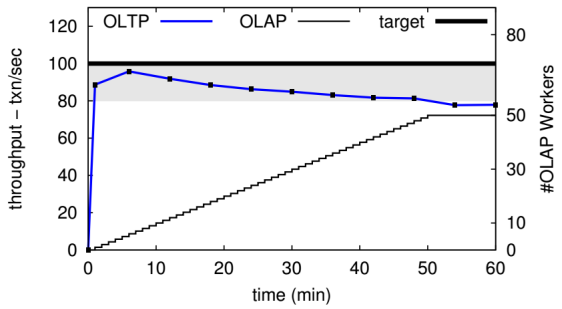
5.2 Scalability of Different Systems Under Test (SUTs)

Rather than comparing quantitatively the different SUTs the authors [CPV+17] have presented scenarios to manifest the expressiveness of the benchmark suite and its metrics. For test, mainly three different SUTs were chosen: OLTP, OLAP, and Hybrid systems. In their analysis they have chosen to set up client balancer and an evaluation period of 60 sec along with regulated OLTP activity by the standard transaction mix in TPC-C. Whereas for the OLAP activity a HTAPBench was set up such that every business query would be selected in accordance to uniform distribution. Through out the experiment, the HTAPBench was configured such that 100 transactions per second were injected. These were equivalent, amount upto 2,099 active OLTP clients and 210 warehouses on a total 117 GB of data. The chosen target transactions per second (tps) was selected as the number of configured warehouses which is over 100 GB dataset. All the experiments are carried out for average of five independent 60 minute runs. Along with this basic set up, based on the type of SUT few changes were done on it.

In this section we display first results published by the authors. Next we report our own limited results, which have faced problems intrinsic with the tool.

5.2.1 Results for OLTP system

A row oriented OLTP engine was deployed and it is configured with enough memory to allow the number of clients required. Also the client balancer within the HTAPBench was configured such that it would consider a default error margin of 20%. Through out the test, it was evident that there exists a declining trend in the OLTP. This depicts serious concerns for the OLAP activities as the engine was able to handle up to 50 OLAP clients.

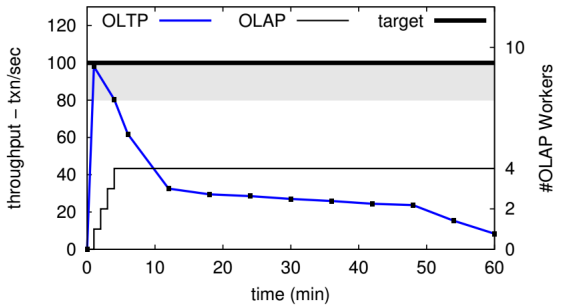


(a) OLTP SUT.

Figure 5.3: OLTP SUT [CPV+17]

5.2.2 Results for OLAP system

Along with the previous setup configuration this experiment deploys a column oriented OLAP engine, in which only the required level of clients allowed is setup. From Figure 5.4 it is clear that this SUT has held the OLTP throughput for a shorter period and this is completely reasonable behavior as the major focus of this engine does not lie with OLTP activity. The client balancer stops to release new OLAP clients as soon as the threshold was broken, that was at the 6th minute. As the threshold breaks even at a lower throughput, the activity of OLTP was kept stable until the end of run time.

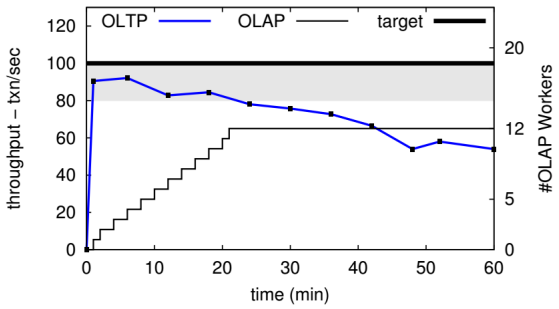


(b) OLAP SUT.

Figure 5.4: OLAP SUT[CPV+17]

5.2.3 Results for Hybrid system

The Figure 2.17 shows that the assigned target was acquired by OLTP in first minute of execution. From then onwards the OLAP streams were deployed by the client balancer until there existed a degrade over OLTP throughput that was beyond the considered error margin. This happened after 20 minutes and hence registering a grand total of 12 OLAP streams. Then begins a slow degrade of the OLTP throughput over the reminder of the test duration.



(c) Hybrid SUT.

Figure 5.5: Hybrid SUT

5.3 Our own Results

From our experimental setup when we analyzed the throughput of OLTP and OLAP, we derived the following results.

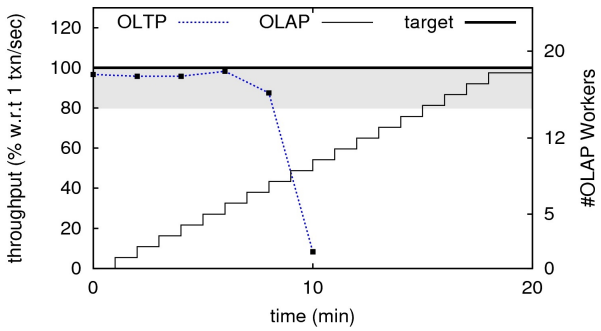


Figure 5.6: Results of HTAPBench over PostgreSQL

From the Figure 5.6 we see that the experiment could hold the OLTP throughput constantly until 7 minutes and then gradually starts decline for a minute and later has a huge rapid decline and ends at 10 minutes. Whereas the OLAP system faces a gradual increase constantly and equals the target point by the end of 19 minutes and stays constant thereafter. The results derive that the system was able to survive for 13.03 *tpmC* and 35.08 *QphH*. Also the total number of OLAP workers until the end were spotted to be 18. Therefore the result of number of QpH for each OLAP worker can be calculated as fraction of QphH and OLAP workers. This derives to be 1.95 QpH. Also the unified matrix Q_pH_pW amounts to 1.95 @ 13.03 *tpmC*.

As Postgres is an OLTP system, to check for validity of our system, we compared our system, its configuration, experimental setup and derived results to the OLTP system experiment done by authors [CPV+17]. We trust that comparing these properties could yield the validity or in case of incorrect results, the reasons or threads that cause the

invalidity could be understood. To start with comparing the *System configuration*, The OLTP system was setup on server with an Intel Xeon E5-2670 v3 CPU with 24 physical cores (48 virtual) at 2.3 GHz with 2 cores and 72GB of memory, running Ubuntu 12.04 LTS as the operating system. Whereas we use Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz, 8GB RAM, 500GB Memory, 64 bits and Linux (Ubuntu 16.4) as operating system.

The authors do not specify any constrain on how long should a OLAP system execute. The lack of *optimal time* where the OLTP system has to run could be serious concern for the validity in replicating their results.

5.4 Summary

In this chapter, we have discussed the state of the art of HTAPBench, an HTAP Benchmark.

We have attempted to talk about various Systems Under Test (SUTs) scalability nature by demonstrating the experiments along with their results that were done in three different scenarios: OLTP, OLAP and Hybrid systems respectively. Additionally, we presented our work i.e., HTAPBench over PostgreSQL which is an OLTP system. This experiment was an attempt to validate the results presented and so was attempted to perform in similar environment. It was not possible for us to repeat the results that authors discuss.

Since the benchmark provides a log of operations, it is possible to employ this for adding freshness measures. This can require the addition of a specific message to indicate the refresh of a replica. Similarly, it can be used for different update schemes. A tool similar to ours, which is capable of calculating Absolute Freshness and Freshness Rate from the logs could be employed to provide these insights. A special extra parameter can be needed to indicate the number of items in the storage.

6. Conclusions and Future Work

6.1 Conclusions

The traditional database workloads, OLTP and OLAP, are used for serving the transactional and decision making applications of organizations respectively. For the purpose of evaluating the system performance, benchmarks have evolved. A Benchmark is a standardized problem or a test which serves as a basis for evaluation or comparisons of various systems. These benchmarks are further classified into different types, based on their evolution target. Transactional Processing Performance Council (TPC) offers different benchmark standards. Transactional processing system benchmarks are especially dedicated to OLTP applications whereas analytical processing system benchmarks are specially dedicated to decision support systems (DSS).

The high demand for evaluating systems that support mixed workloads (HTAP) necessitates the evolution of mixed workload benchmarks, which are crucial to help these technologies developed by enabling the comparison of systems, helping the community to realize which design features are useful and which are not. In spite of a dramatic growth in HTAP systems, there is research gap in standard HTAP benchmarks and there is no standard available in the market.

However some HTAP benchmarks exist, which consider mixed workloads and the system-specific aspects of HTAP systems, like CH-benCHmark, HTAPBench and CBTR. The major challenges that these HTAP systems face are to evaluate performance, freshness, flexibility, isolation and elasticity issues.

These mixed workload benchmarks do not satisfy all the requirements, mainly the challenge of providing a standard and informative measure for freshness. To understand this challenge, we have studied the different freshness metrics provided by various authors. In our work we analyzed various freshness metrics, as they concern to evaluating HTAP systems. The common metrics proposed to represent the freshness of the data include

staleness, age, timeliness, freshness, currency, obsolescence, up-to-datedness etc. In various scenarios such as data integration or dynamic web database, continuous update of data plays a key role, hence the freshness of the data as a metric (data quality) plays a vital role for understanding the performance of a system. Without such a measure HTAP tools can fool a benchmarking tool by simply reducing freshness to achieve a higher throughput.

Evaluating freshness is also a potential area of interest in designing HTAP systems, since the query engine might be able to leverage this information in its processing, deciding, for example if a given optimization is possible by reducing freshness slightly while keeping tight SLAs with users. It can also be used for co-processor accelerated operations, which face the same problem of deciding if it is possible to run an operator over a replica instead than over the live data.

We reviewed freshness metrics proposed in the literature, and we employed the following setup to evaluate the applicability of these metrics:

6.1.1 Tests

The YCSB benchmark was built on Java and it possesses extensibility such that the YCSB clients can be modified according to user requirements. This benchmark workload has combination reads and updates and these mixed workloads run simultaneously, hence we have chosen YCSB Benchmark in our environment as the Benchmark to perform tests over our prototype database.

It should be noted that the read components of YCSB are OLTP and, hence, it cannot be considered an HTAP benchmark. Nonetheless we deemed it fit to our study since it combines reads and updates following statistical distributions representative of high demand systems (namely a Zipf distribution of requests, with about 80% cold items and 20% hot items.

6.1.1.1 Implementation

For the task of analyzing freshness in an HTAP system we used a prototype database which was built based on the architecture of Octopus DB. Our prototype database was called Blinktopus DB, and it was developed by students in our university. In this Thesis we modified this system according to the YCSB benchmark to run the test (i.e we exposed the expected endpoints such that it could be connected to YCSB). We also added the complete functionality of concurrency control with locking, and the concept for a periodic refresh of the materialized views based on a provided refresh rate.

Our tests using a YCSB mixed workload were run in every 3, 30 and 300 second intervals. In these tests we observed that the throughput of the system at 300 seconds is higher and decreases when the time interval decreases. Hence we confirm experimentally the case of higher freshness leading to lower throughput. We have tracked the complete results into a file. The freshness measurements are performed off line on the stored data file.

6.1.2 Evaluation

Through our study calculating these measurements we note the pros and cons of the measures in describing the quality of individual read operations and of the read replica as a whole.

We note that metrics can be classified according to their use of information posterior to the update time of replicas (i.e., live information about the updates that are still not visible to the replica). The distinction between measures into these two large groups is important because it establishes the way in which these measurements can be applied and the strategies that can be adopted to model per-item freshness. Measurements that use live information are suitable for adoption in benchmarks. They can be computed after the execution of the tests, by analyzing the logs. Furthermore they provide precise insights (not based on estimations), such as the case of the Absolute Freshness and Freshness Rate, which count as fresh exactly those items that have been updated on the live system.

Measures that do not use live information are the most suitable to live systems when there is limited observability. These measures are to be preferred for HTAP designs that might employ freshness measurements as part of the online statistics used by query optimizers.

We also note that some measures like the Shifting Window Staleness are not pertinent to our use case, as they does not fall into any of the two large groups we have discussed, and hence it is unclear what are its real-world application.

We have also noted that global measures and time-dependent measures which disregard the volatility of items provide insufficient information for our use case. Among the first group they fail to give insights into the state of the read replica, or into the freshness of specific read requests. Among the second, they are not adequate for a write-intensive scenario when the number of updates is more informative than the time elapsed. In the same line of thoughts we have noted that sophisticated approaches, like the Exponential Smoothing Staleness, though potentially useful, with its difficulties for aggregation fall short when compared to Absolute Timeliness.

We identified Absolute Freshness and Freshness Rate as the preferred metrics for measures that require information posterior to the update of the replica. We also proposed Absolute Timeliness as the best metric for measures which are tailored to cases where there is no access to information posterior to the update time of the replica. In addition, we invented and tested a novel measure, called Timeliness Rate so as to model the Freshness Rate in the latter case. We show that our metric is useful and readily applicable.

We concluded our discussion by observing that some measurements, by themselves, are able to answer (with varying degrees of goodness) the information need of evaluating the quality of read transactions (e.g. Absolute Freshness, Absolute Timeliness) and the quality of the read replica (e.g. Freshness Rate). Other measures serve only as

building blocks to fulfill these and other information needs. As a result from our study, though we recommend the adoption of some metrics we do not discourage the use of others which could be applied, if they can be helpful to fulfill other information needs apart from those considered for our study. We envision as future work a system which will offer users these metrics such that they can employ them as building blocks to create dashboards and understand better their systems with regards to this data quality dimension.

6.2 Future Work

We have tested the HTAPBench benchmark using OLTP, OLAP and Hybrid workloads. Postgres database has been connected with this benchmark and tested. As a future work, other databases (NuoDB, SAP HANA etc.) can also be connected with this benchmark and the performance of those databases can be analyzed. This helps not only to understand the benchmark credibility, in addition databases performance can be more clearly understood. Also as future work, there needs to be more tests to validate that the benchmark can be used by any user to reproduce the results published by authors.

We suggest that important future work is to use our analysis framework on different HTAP systems, and to extend it for two possible cases: a more fine-grained notion of refresh rates (i.e. with it being applied to a subsection of the read replica), and the case when requests themselves are able to introduce their freshness requirements (as we saw is applied in Google Cloud Spanner).

6.3 Concluding Remarks

We expect our work to contribute to the development of standards for measuring freshness, helping in practical evaluations of this data quality dimension, and helping to build query engines able to reason about freshness in any situation where replicated data management is needed, like HTAP systems or co-processor accelerated systems.

Bibliography

[AIA14] Ioannis Alagiannis, Stratos Idreos, and Anastasia Ailamaki. H2o: a hands-free adaptive store. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1103–1114. ACM, 2014. (cited on Page 13)

[APM16] Joy Arulraj, Andrew Pavlo, and Prashanth Menon. Bridging the archipelago between row-stores and column-stores for hybrid workloads. In *Proceedings of the 2016 International Conference on Management of Data*, pages 583–598. ACM, 2016. (cited on Page 13)

[BBB+17] David F. Bacon, Nathan Bales, Nico Bruno, Brian F. Cooper, Adam Dickinson, Andrew Fikes, Campbell Fraser, Andrey Gubarev, Milind Joshi, Eugene Kogan, Alexander Lloyd, Sergey Melnik, Rajesh Rao, David Shue, Christopher Taylor, Marcel van der Holst, and Dale Woodford. Spanner: Becoming a sql system. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pages 331–343, New York, NY, USA, 2017. ACM. (cited on Page 3, 7, 14, and 39)

[BBF15] Melyssa Barata, Jorge Bernardino, and Pedro Furtado. An overview of decision support benchmarks: Tpc-ds, TPC-H and SSB. In *New Contributions in Information Systems and Technologies - Volume 1 [WorldCIST'15, Azores, Portugal, April 1-3, 2015].*, pages 619–628, 2015. (cited on Page 28)

[BC05] Brian Babcock and Surajit Chaudhuri. Towards a robust query optimizer: a principled and practical approach. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 119–130. ACM, 2005. (cited on Page 42)

[BDM+16] Alexander Böhm, Jens Dittrich, Niloy Mukherjee, Ippokratis Pandis, and Rajkumar Sen. Operational analytics data management systems. *Proceedings of the VLDB Endowment*, 9(13):1601–1604, 2016. (cited on Page 7 and 9)

[BFG+06] Philip A Bernstein, Alan Fekete, Hongfei Guo, Raghu Ramakrishnan, and Pradeep Tamma. Relaxed-currency serializability for middle-tier caching

and replication. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 599–610. ACM, 2006. (cited on Page 2, 3, 7, and 38)

[Bog12] Anja Bog. Benchmarking composite transaction and analytical processing systems: the creation of a mixed workload benchmark and its application in evaluating the impact of database schema optimizations in mixed workload scenarios. 2012. (cited on Page 2, 7, 22, and 23)

[Bou04] Mokrane Bouzeghoub. A framework for analysis of data freshness. In *Proceedings of the 2004 international workshop on Information quality in information systems*, pages 59–67. ACM, 2004. (cited on Page 3, 7, 35, 36, 48, and 49)

[BPZ11] Anja Bog, Hasso Plattner, and Alexander Zeier. A mixed transaction processing and operational reporting benchmark. *Information Systems Frontiers*, 13(3):321–335, 2011. (cited on Page 2, 3, and 54)

[BSM03] Matthew Bovee, Rajendra P Srivastava, and Brenda Mak. A conceptual framework and belief-function approach to assessing overall information quality. *International journal of intelligent systems*, 18(1):51–74, 2003. (cited on Page 48 and 49)

[BT12] Benchmarks and Performance Tests. Chapter 7. Retrieved from http://catalogue.pearsoned.co.uk/samplechapter/0130659037.pdf, 2012. Last accessed 31 December 2017. (cited on Page 25)

[BUU17] Massachusetts Institute of Technology Brown University, Carnegie Mellon University and Yale University. H-store. Retrieved from http://hstore.cs.brown.edu/wordpress/wp-content/uploads/2011/05/tpce.png, 2017. Last accessed 12 December 2017. (cited on Page xiii and 25)

[BWPT98] Donald Ballou, Richard Wang, Harold Pazer, and Giri Kumar Tayi. Modeling information manufacturing systems to determine information product quality. *Management Science*, 44(4):462–484, 1998. (cited on Page 48 and 50)

[CAA+10] Shimin Chen, Anastasia Ailamaki, Manos Athanassoulis, Phillip B. Gibbons, Ryan Johnson, Ippokratis Pandis, and Radu Stoica. TPC-E vs. TPC-C: characterizing the new TPC-E benchmark via an I/O comparison study. *SIGMOD Record*, 39(3):5–10, 2010. (cited on Page 25)

[CCG+11] Yu Cao, Chun Chen, Fei Guo, Dawei Jiang, Yuting Lin, Beng Chin Ooi, Hoang Tam Vo, Sai Wu, and Quanqing Xu. Es 2: A cloud data storage system for supporting both oltp and olap. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 291–302. IEEE, 2011. (cited on Page 13)

[CCH+14] Henggang Cui, James Cipar, Qirong Ho, Jin Kyu Kim, Seunghak Lee, Abhimanu Kumar, Jinliang Wei, Wei Dai, Gregory R Ganger, Phillip B Gibbons, et al. Exploiting bounded staleness to speed up big data analytics. In *USENIX Annual Technical Conference*, pages 37–48, 2014. (cited on Page 2, 3, and 7)

[CFG+11] Richard Cole, Florian Funke, Leo Giakoumakis, Wey Guy, Alfons Kemper, Stefan Krompass, Harumi Kuno, Raghunath Nambiar, Thomas Neumann, Meikel Poess, et al. The mixed workload ch-benchmark. In *Proceedings of the Fourth International Workshop on Testing Database Systems*, page 8. ACM, 2011. (cited on Page 2, 3, 7, 90, and 91)

[CGHJ12] Graham Cormode, Minos Garofalakis, Peter J Haas, and Chris Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*, 4(1–3):1–294, 2012. (cited on Page 42)

[CGM99] Junghoo Cho and Hector Garcia-Molina. The evolution of the web and implications for an incremental crawler. Technical report, Stanford, 1999. (cited on Page 49)

[CGM00] Junghoo Cho and Hector Garcia-Molina. Synchronizing a database to improve freshness. In *ACM sigmod record*, volume 29, pages 117–128. ACM, 2000. (cited on Page 48 and 49)

[CGM03] Junghoo Cho and Hector Garcia-Molina. Effective page refresh policies for web crawlers. *ACM Transactions on Database Systems (TODS)*, 28(4):390–426, 2003. (cited on Page 36, 38, 48, and 49)

[CJW+16] Jack Chen, Samir Jindel, Robert Walzer, Rajkumar Sen, Nika Jimsheleishvilli, and Michael Andrews. The memsql query optimizer: A modern optimizer for real-time analytics in a distributed database. *Proc. VLDB Endow.*, 9(13):1401–1412, September 2016. (cited on Page 13)

[Cor12] Standard Performance Evaluation Corporation. Corporation website. Retrieved from http://www.spec.org/, 2012. Last accessed 25 December 2017. (cited on Page 22)

[Cou17a] Transaction Processing Performance Council. Council website. Retrieved from http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-c_v5.11.0.pdf, 2017. Last accessed 12 December 2017. (cited on Page xiii and 24)

[Cou17b] Transaction Processing Performance Council. Council website. Retrieved from http://www.tpc.org/, 2017. Last accessed 12 December 2017. (cited on Page 22, 23, and 26)

[CPV+17] Fábio Coelho, João Paulo, Ricardo Vilaça, José Pereira, and Rui Oliveira. Htapbench: Hybrid transactional and analytical processing benchmark. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*, ICPE '17, pages 293–304, New York, NY, USA, 2017. ACM. (cited on Page xiii, xvi, 2, 3, 7, 34, 35, 90, 91, 92, 93, and 94)

[CST+10] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154. ACM, 2010. (cited on Page xiii, 27, and 28)

[DJ11] Jens Dittrich and Alekh Jindal. Towards a one size fits all database architecture. In *CIDR*, 2011. (cited on Page 12, 41, and 46)

[ES07] Adir Even and Ganesan Shankaranarayanan. Utility-driven assessment of data quality. *ACM SIGMIS Database: the DATABASE for Advances in Information Systems*, 38(2):75–93, 2007. (cited on Page 48 and 49)

[FCP+12] Franz Färber, Sang Kyun Cha, Jürgen Primsch, Christof Bornhövd, Stefan Sigg, and Wolfgang Lehner. Sap hana database: data management for modern business applications. *ACM Sigmod Record*, 40(4):45–51, 2012. (cited on Page 7 and 12)

[FK09] Daniela Florescu and Donald Kossmann. Rethinking cost and performance of database systems. *SIGMOD Rec.*, 38(1):43–48, June 2009. (cited on Page 14)

[FKN11] Florian Funke, Alfons Kemper, and Thomas Neumann. Benchmarking hybrid oltp and olap database systems. In *In BTW*, pages 390–409, 2011. (cited on Page xiii, 28, and 33)

[Gar14] Gartner. Hybrid transaction/analytical processing will foster opportunities for dramatic business innovation. Retrieved from https://www.gartner.com/imagesrv/media-products/pdf/Kx/KX-1-3CZ44RH.pdf, 2014. Last accessed 27 December 2017. (cited on Page 1, 9, and 32)

[GJS09] Lukasz Golab, Theodore Johnson, and Vladislav Shkapenyuk. Scheduling updates in a real-time stream warehouse. In *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*, pages 1207–1210. IEEE, 2009. (cited on Page 38, 48, and 49)

[GKP+10] Martin Grund, Jens Krüger, Hasso Plattner, Alexander Zeier, Philippe Cudre-Mauroux, and Samuel Madden. Hyrise: a main memory hybrid storage engine. *Proceedings of the VLDB Endowment*, 4(2):105–116, 2010. (cited on Page xiii, 7, 11, and 12)

[GLR05] Hongfei Guo, Per-Åke Larson, and Raghu Ramakrishnan. Caching with good enough currency, consistency, and completeness. In *Proceedings of the 31st international conference on Very large data bases*, pages 457–468. VLDB Endowment, 2005. (cited on Page 38 and 48)

[GLRG04] Hongfei Guo, Per-Åke Larson, Raghu Ramakrishnan, and Jonathan Goldstein. Relaxed currency and consistency: how to say good enough in sql. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 815–826. ACM, 2004. (cited on Page 3, 7, 37, and 38)

[Gra91] Jim Gray, editor. *The Benchmark Handbook for Database and Transaction Systems (1st Edition)*. Morgan Kaufmann, 1991. (cited on Page 22 and 23)

[Hin02] Holger Hinrichs. *Datenqualitätsmanagement in Data Warehouse-Systemen*. PhD thesis, Universität Oldenburg, 2002. (cited on Page 50)

[HKK09] Bernd Heinrich, Mathias Klier, and Marcus Kaiser. A procedure to develop metrics for currency and its application in crm. *Journal of Data and Information Quality (JDIQ)*, 1(1):5, 2009. (cited on Page 3, 7, 37, and 38)

[IGN+12] Stratos Idreos, Fabian Groffen, Niels Nes, Stefan Manegold, K. Sjoerd Mullender, and Martin L. Kersten. Monetdb: Two decades of research in column-oriented database architectures. *IEEE Data Eng. Bull.*, 35(1):40–45, 2012. (cited on Page 8 and 12)

[KKN+08] Robert Kallman, Hideaki Kimura, Jonathan Natkins, Andrew Pavlo, Alexander Rasin, Stanley Zdonik, Evan P. C. Jones, Samuel Madden, Michael Stonebraker, Yang Zhang, John Hugg, and Daniel J. Abadi. H-store: A high-performance, distributed main memory transaction processing system. *Proc. VLDB Endow.*, 1(2):1496–1499, August 2008. (cited on Page 8)

[KN11] Alfons Kemper and Thomas Neumann. Hyper: A hybrid oltp&olap main memory database system based on virtual memory snapshots. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 195–206. IEEE, 2011. (cited on Page xiii, 7, and 11)

[KNF+12] Alfons Kemper, Thomas Neumann, Florian Funke, Viktor Leis, and Henrik Mühe. Hyper: Adapting columnar main-memory data management for transactional and query processing. *IEEE Data Eng. Bull.*, 35:46–51, 2012. (cited on Page 3, 7, and 11)

[LCC+15] Tirthankar Lahiri, Shasank Chavan, Maria Colgan, Dinesh Das, Amit Ganesh, Mike Gleeson, Sanket Hase, Allison Holloway, Jesse Kamp, Teck-Hua Lee, et al. Oracle database in-memory: A dual format in-memory database. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, pages 1253–1258. IEEE, 2015. (cited on Page 13)

[LMHK+17] Juchang Lee, SeungHyun Moon, Kyu Hwan Kim, Deok Hoe Kim, Sang Cha, and Wook-Shin Han. Parallel replication across formats in sap hana for scaling out mixed oltp/olap workloads. 10:1598–1609, 08 2017. (cited on Page 3, 7, and 39)

[LMTdU08] Guzmán Llambías, Regina Motz, Federico Toledo, and Simon de Uvarow. Learning to get the value of quality from web data. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 1018–1025. Springer, 2008. (cited on Page 50)

[LR04] Alexandros Labrinidis and Nick Roussopoulos. Exploring the tradeoff between performance and data freshness in database-driven web servers. *The VLDB Journal—The International Journal on Very Large Data Bases*, 13(3):240–255, 2004. (cited on Page 36 and 38)

[MGBA17] Darko Makreshanski, Jana Giceva, Claude Barthels, and Gustavo Alonso. Batchdb: Efficient isolated execution of hybrid oltp+ olap workloads for interactive applications. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 37–50. ACM, 2017. (cited on Page 7, 14, 21, and 37)

[NLF05] Felix Naumann, Ulf Leser, and Johann Christoph Freytag. *Quality-driven integration of heterogeneous information systems*. Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät II, Institut für Informatik, 2005. (cited on Page 3, 7, 37, 38, and 49)

[OC12] Paolo Bouquet Oleksiy Chayka, Themis Palpanas. Defining and measuring data-driven quality dimension of staleness. *DISI - Via Sommarive 14-38123 Povo-Trento (Italy)*, 20(S2), 2012. (cited on Page 3, 7, and 50)

[OTT17] Fatma Özcan, Yuanyuan Tian, and Pinar Tözün. Hybrid transactional/analytical processing: A survey. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pages 1771–1775, New York, NY, USA, 2017. ACM. (cited on Page 7, 9, 14, and 20)

[Pla09] Hasso Plattner. A common database approach for oltp and olap using an in-memory column database. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, pages 1–2, New York, NY, USA, 2009. ACM. (cited on Page 9)

[PS04] Meikel Poess and John M. Stephens, Jr. Generating thousand benchmark queries in seconds. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, pages 1045–1053. VLDB Endowment, 2004. (cited on Page 31)

[PSKL02] Meikel Pöss, Bryan Smith, Lubor Kollár, and Per-Åke Larson. Tpc-ds, taking decision support benchmarking to the next level. In Michael J. Franklin,

Bongki Moon, and Anastassia Ailamaki, editors, *SIGMOD Conference*, pages 582–587. ACM, 2002. (cited on Page 31)

[PWM+14] Iraklis Psaroudakis, Florian Wolf, Norman May, Thomas Neumann, Alexander Böhm, Anastasia Ailamaki, and Kai-Uwe Sattler. Scaling up mixed workloads: a battle of data freshness, flexibility, and scheduling. 2014. Published by: Springer International Publishing AG. Benchmark source code can be found in the referenced URL. (cited on Page xiii, 2, 7, 15, 16, 17, 18, 19, 20, and 54)

[QL07] Huiming Qu and Alexandros Labrinidis. Preference-aware query and update scheduling in web-databases. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 356–365. IEEE, 2007. (cited on Page 3, 7, 36, 48, and 49)

[RBSS02] Uwe Röhm, Klemens Böhm, Hans-Jörg Schek, and Heiko Schuldt. Fas—a freshness-sensitive coordination middleware for a cluster of olap components. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*, pages 754–765. Elsevier, 2002. (cited on Page 3, 7, 38, and 63)

[RDS02] Ravishankar Ramamurthy, David J. DeWitt, and Qi Su. Chapter 38 - a case for fractured mirrors. In Philip A. Bernstein, , Yannis E. Ioannidis, , Raghu Ramakrishnan, , and Dimitris Papadias, editors, {*VLDB*} *'02: Proceedings of the 28th International Conference on Very Large Databases*, pages 430 – 441. Morgan Kaufmann, San Francisco, 2002. (cited on Page 37)

[Sen18] Ramachandra Sen. Characterizing resource sensitivity of database workloads,, 2018. (cited on Page 21)

[Web18] Web. tps://www.softwareadvice.com/resources/wp-content/uploads/OLAP-alternative.png, 2018. (cited on Page xiii and 1)

[XHLC08] Ming Xiong, Song Han, Kam-Yiu Lam, and Deji Chen. Deferrable scheduling for maintaining real-time data freshness: Algorithms, analysis, and results. *IEEE Transactions on Computers*, 57(7):952–964, 2008. (cited on Page 38)