Effects of Virtualization in Database Benchmarking

by
Jigarkumar Jayantibhai Rathod

A thesis submitted to the School of Computing
Florida Institute of Technology
in partial fulfillment of the
requirements for the degree of

Master of Science
in
Computer Science

Melbourne, Florida
May 2017
We, the undersigned committee, hereby approve the attached thesis.

**Effects of Virtualization in Database Benchmarking**

by

Jigarkumar Jayantibhai Rathod

Marco Carvalho, Ph.D.
Major Advisor
Executive Director, Harris Institute for Assured Information
Professor and Department Head, School of Computing

Marius Silaghi, Ph.D.
Committee Member
Associate Professor, Computer Sciences and Cybersecurity

Muzaffar Shaikh, Ph.D.
Committee Member
Department Head, Engineering Systems
Abstract

Many applications are being deployed on the cloud because virtualization provides inexpensive virtual machines which are easy to create and use. To perform benchmark in scale, one has to rely on virtualization. Virtualization itself, however affects the performance of the database systems. In this thesis, we investigate the unanticipated effects of the virtualization on the database systems by running experiments on virtual machines. The results of our experiments indicate that the performance of a virtual machine is highly influenced by the host machine.
# Contents

1 Introduction 1

2 Literature Review 5

2.1 Types of Database Systems ................................. 5

2.2 Review of Benchmarking ................................. 8

2.2.1 Benchmarking SQL Database ................................. 8

2.2.2 Benchmarking NoSQL database ...................... 11

2.3 Tools used for Benchmarking ................................. 15

2.4 Virtualization Environment ................................. 19

3 Proposed Approach 22

3.1 Our Approach ........................................ 22

3.2 Parameters that affect Benchmarking ...................... 23
4 Experimental Design and Implementation

4.1 Yahoo Cloud Serving Benchmark (YCSB) ............................................. 26

4.2 Implementation of Java program ............................................................. 27

4.3 MySQL database ................................................................................. 27

4.4 Tableau ............................................................................................... 28

4.5 Virtual Infrastructure Network Emulator (VINE) ................................. 28

5 Experiments and Analysis

5.1 Identify the effect of time ................................................................. 31

5.2 Identify the effect of running stress tests with different number of threads ................................................................. 32

5.3 Comparing the effects of multiple threads ........................................... 33

5.4 The effects of multiple threads for a different VM configuration ........... 34

5.5 Running tests on multiple VMs in sequential manner .......................... 35

5.6 Running tests on multiple VMs in a parallel manner ............................ 39

6 Recommendations .................................................................................. 43

7 Conclusion ............................................................................................. 44
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Our approach to design the experiments</td>
<td>22</td>
</tr>
<tr>
<td>3.2</td>
<td>3 Tier Architecture of the modern applications</td>
<td>24</td>
</tr>
<tr>
<td>4.1</td>
<td>A high level design diagram of the environment used to perform experiments</td>
<td>27</td>
</tr>
<tr>
<td>4.2</td>
<td>A sample testbed which has 7 MySQL servers connected with 7 YCSB clients</td>
<td>30</td>
</tr>
<tr>
<td>5.1</td>
<td>Schematic representation of the Experiment</td>
<td>32</td>
</tr>
<tr>
<td>5.2</td>
<td>Throughput vs. Time and latency vs. time diagram for the results of stress test run with 25 threads</td>
<td>33</td>
</tr>
<tr>
<td>5.3</td>
<td>Time to complete stress test to insert 100,000 records for each thread configuration</td>
<td>34</td>
</tr>
<tr>
<td>5.4</td>
<td>Average throughput vs. time and average latency vs. time result colored by the thread configuration</td>
<td>35</td>
</tr>
</tbody>
</table>
5.5 Throughput vs. time for 20, 25, 30 and 35 threads colored by individual runs. .................................................. 35

5.6 Throughput vs. time for 20, 25, 30 and 35 threads colored by individual runs performed on MySQL database installed on small image. .................................................. 36

5.7 Throughput vs. time for 20, 25, 30 and 35 threads colored by individual runs performed on MySQL database installed on small image. .................................................. 38

5.8 Graph representing the throughput equations of each VM. ........ 39

5.9 The placement of VM on physical server. ............................... 40

5.10 Running stress tests in parallel. ........................................... 41
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Equation to represent the throughput pattern of different thread configuration on micro image.</td>
<td>36</td>
</tr>
<tr>
<td>5.2</td>
<td>The result of T-tests for the experiment performed on the micro image.</td>
<td>36</td>
</tr>
<tr>
<td>5.3</td>
<td>Equation representing the throughput pattern of different thread configuration on small image.</td>
<td>37</td>
</tr>
<tr>
<td>5.4</td>
<td>The result of T-tests for the experiment performed on the small image.</td>
<td>37</td>
</tr>
<tr>
<td>5.5</td>
<td>Equation to represent the throughput pattern of different VM.</td>
<td>38</td>
</tr>
<tr>
<td>5.6</td>
<td>The result of T-tests for the experiment performed with 5 small VMs running the MySQL service.</td>
<td>40</td>
</tr>
</tbody>
</table>
Acknowledgements

My brother and sister-in-law, Mirav and Neha Rathod, have motivated me to undertake challenging tasks. I appreciate their love and support.

I would like to express my sincere gratitude to my advisor Prof. Marco Carvalho for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor.

A special thanks to my pals Tapas Joshi, Prassanna Rao, Rohit Kadam, Pritam Chakraborty, Siddhesh Jethe, Vishrut Majmudar, and Aditya Karanjkar who helped me to stay on track.
This thesis is dedicated to my parents Jayantibhai and Ilaben Rathod

whose love and advice

have meant so much to me.
Chapter 1

Introduction

Cloud services alleviate the pain of provisioning and maintenance that comes with physical servers, and provide inexpensive virtual machines which can be created, deleted, migrated and used as one sees fit. These services also reduce the upfront costs associated with the servers and has the following characteristics [31]:

- On-demand self-service which allows the consumer to unilaterally provision resources as they need.
- Using ubiquitous network access, a user can connect to their resources using any device.
- Resource pooling allows the cloud provider to utilize the same physical resources among multiple customers.
- Rapid elasticity equips the application to scale outward or inward as requested.
- Pay-per-use model charges users only for the resources they used. This model is beneficial for small and medium sized organizations.

The majority of high performing clouds use virtualization for server consolidation. Hardware, software, memory, storage and network are few of the several types of
virtualization. Server virtualization creates multiple logically isolated virtual machines running the different operating systems on a physical server.

Identifying what kind of applications are suitable for the cloud has been an area of research. Abadi [2] reviewed transactional and analytical data management applications to determine their suitability in the cloud. Transactional applications are write-intensive and require Atomicity, Consistency, Integrity and Durability (ACID) guarantees whereas analytical applications are read-intensive. Hence, analytical applications can capitalize cloud services. On the other hand, Bose et al. [7] performed experiments in virtual environment using a non-comparable implementation of TPC-C and TPC-E benchmarks. Results of experiments indicate that the virtual machines were able to provide more than 85% of the throughput of the native machine. Bose et al. concluded that virtualization technologies can efficiently cater transactional workload.

451 Research’s [1] the Voice of the Enterprise[1] conducted a survey to identify the adoption of Cloud Computing in the enterprises. Their survey shows that about 67% respondents are currently using a non-cloud environment, the rest of the respondents are using cloud solutions in the form of Software-as-a-Service, Infrastructure-as-a-Service, on-premise private cloud and hosted private cloud. The major obstacles to cloud adoption are security and performance. Many respondents believe that as more enterprises adapt to the cloud environment, the concerns regarding security will diminish. About 55% of respondents expect to utilize cloud services within the next two years. The results of the survey indicates that the cloud services will continue to grow in the foreseeable future.

In order to maximize the performance of the application, application servers and database servers are usually put together. Currently, there are three ways to use database systems in the cloud [16]:

1. Cloud native relational DBMS eases the task of infrastructure management and allows one to easily scale up via application programming interfaces

---


3. Cloud capable relational/non-relational database using virtual machine require the installation of the database server. This approach gives full control over database systems. MySQL, Oracle, PostgreSQL are cloud capable relational databases. Apache Cassandra, MongoDB, Voldemort are cloud capable non-relational databases.

Modern application expects huge traffic. To measure the efficiency of such applications, one need to perform benchmark in scale. Virtualization is the ideal choice to do so, as it provides on demand scalability. However, virtualization itself affects the performance of the VM. In this thesis, we used database benchmark to investigate into the unanticipated effects of virtualization. Understanding the effects of virtualization helps an organization to provide realistic guarantees if their application is deployed in virtual environment.

Usually, database are compared using aggregate measures such as overall throughput and average latency. As VMs exhibit consistent pattern for throughput, we define throughput as a function of time and use it for comparison.

The main contributions of this thesis are as following:

- We reviewed prior approaches to benchmarking the database systems.

[^2]: https://aws.amazon.com/rds/
[^4]: https://www.enterprisedb.com/
[^5]: https://cloud.google.com/sql/docs/
[^6]: https://www.mongodb.com/cloud/atlas
[^7]: https://redislabs.com/
[^8]: https://aws.amazon.com/simpledb/
• We use database benchmarks to study the effects of virtualization on database performance.

The rest of this thesis is organized as follows: In Chapter 2, we briefly look into the different types of the database systems and their design principle. It covers different benchmarking approaches. Additionally, we reviewed the benchmarking tools and the virtualization platforms that are currently being used in the industry. To identify the effects of virtualization it is very important to design experiments. In Chapter 3 we outline the approach we used to design experiments. In Chapter 4 we specify the tools and frameworks that we used for experimentation. In Chapter 5 we describe the environment that we created to perform the experiments and analyze the effects of the virtual environment. Possible direction of future work is described in Chapter 6. Chapter 7 summarizes and concludes our findings.
Chapter 2

Literature Review

In this chapter, we will look into the different types of database systems and their design principles. We will review the approaches and tools to benchmark the database systems. For all the tools, we summarize their origin, purpose and development status. We also review the virtualization environments that are popular in the industry.

2.1 Types of Database Systems

The database system is the most important part of any application. The database systems are selected according to the requirements of application. For example, some applications require 24*7 availability whereas others need strong consistency. To cater such varied requirements, different database solutions exist. In a nutshell, database systems can be categorized as following.

- **SQL database**: In 1970, E. F. Codd proposed a relational model of data which laid the foundation for the SQL database [25]. With time, many SQL database management systems have emerged such as Oracle, MySQL, Postgres etc.
- **NoSQL database**: All the databases who do not follow relational data models are referred as NoSQL databases. MongoDB, Cassandra, Voldemort are few of the example of NoSQL database.

To cater the requirement of modern applications, one has to consider Distributed Databases. Eric Brewer outlined the CAP theorem [13] which gives an idea on the design of distributed databases. The CAP theorem states that any distributed system with multiple networks can have at most two of three desirable properties:

- **Consistency** - System is up-to-date and serves the most recent information.
- **Availability** - System is available all the time.
- **Partition** - Despite failure of network devices, systems continues to perform operations.

SQL databases follow the ACID principle [25]. ACID stands for: Atomicity - Either transactions are completed entirely, or no changes happen at all. Consistency - It ensures that database remains valid after the execution of the transaction. Isolation - Concurrently executing transactions do not interfere with each other. Durability - Database should persist the changes once the transaction is completed. Using the ACID principle, SQL databases assure that all transactions have been processed reliably.

SQL databases are not the ideal choice for big data because of the following reasons [27].

- **Performance**: As data grows over time, performance of SQL databases degrades. In such a case, one has to either shard the data to partition it across different sites or abandon the SQL database.
- **Flexibility**: SQL databases works on structured data. Hence, to add a new piece of information, one needs to redesign the table structure.
Dan [20] proposed an alternative to traditional transactional model called BASE (basically available, soft state, eventually consistent). Traditional databases perform two-phase commits (2PC) to guarantee consistency in a Distributed Environment. In Phase-1, all the databases involved in the transaction precommit the transaction and notify the transaction coordinator about the status of the transaction. If the involved databases agree for the transaction, then in phase-2, the transaction coordinator asks each database to commit the data. Dan showed a sample implementation of BASE by using a message queue. Transactions are decoupled according to functional groups. Any changes to the data are pushed in the queue. A separate message processing component dequeues the message and performs the transaction. BASE supports partial failures to ensure higher availability which means that if a database is partitioned across five database servers, then failure in a database server will impact only 20% of the users.

NoSQL databases follow BASE and provide high availability by relaxing consistency. Based on their data model, NoSQL databases can be categorized as following [15].

- **Key Value Pair**: The key value pair data model is inspired from the key-value data structure. The applications which perform read-heavy tasks benefit from it. Inconsistency for read operations are accepted for key-value databases. It is the application program or client’s job to resolve such inconsistencies.

- **Column Family**: These kind of databases focus on columns. In this data model, a key consists of 3 parts: name, value, and timestamp. Timestamp helps to get the latest information about the data. They are useful for highly sparse data collection.

- **Document**: A document database stores values in a document like structure such as JSON or XML.

- **Graph Database**: In this kind of database, graphs are used to represent the schema. Graph database consists of Nodes, Edges, and Properties.

Twelve years later, Eric Brewer pointed out that the CAP theorem was misunderstood [8]. In case of partitions, most of the designers sacrifice consistency to
provide availability. However, the goal of the CAP theorem is to maximize consistency and availability in case of partition.

2.2 Review of Benchmarking

Database systems are benchmarked to identify the cause of performance bottlenecks, collect performance metrics and compare them with real-world requirements and identify alternative options by comparing different database systems. Jim Gray [14] pointed out four important characteristics of a useful benchmark.

• Relevant: A benchmark should measure the performance for the relevant application domain.

• Portable: Benchmark should be easy to implement on different systems.

• Scalable: To efficiently benchmark larger system, the benchmark should be able to scale up.

• Simple: The benchmark itself and results produced must be understandable.

2.2.1 Benchmarking SQL Database

Wisconsin Benchmark (1983)

Wisconsin Benchmark [6] is one of the earlier approach to benchmarking the database systems. University-INGRES, Commercial-INGRES, Oracle, The IDM/500 Database Machine and The Database Machine DIRECT were compared using the wisconsin benchmark.

To perform the benchmark, four relations were defined namely thoustup, twothoustup, fivethoustup and tenthoustup which contain 1000, 2000, 5000 and 10000 tuples respectively. Each relation has three string attributes of 52 characters and five integer attributes whose value ranges from 0-9999.
The Wisconsin benchmark uses a random generator to create uniformly distributed values. A standard set of queries were developed for relational operations such as Selection, Projection, Joins, Aggregates, and Update. Total time to complete operation is the primary measure of the Wisconsin Benchmark. The benchmark is criticized for being single user.

**A Measure of Transaction Processing Power (1985)**

Many database vendors started to quote their database’s ability to process high numbers of transactions in a second. Bitton et al. [5] identified that as transactions are not standardized, one cannot compare them. Hence, they defined standard operations to measure the performance of online transaction processing systems (OLTP).

They defined three standard operations. Debit Credit operation - A transaction to debit a bank account, perform double entry bookkeeping and then respond to the client. A branch, a teller working at the branch and an account is picked randomly to perform the debit-credit operation. The scan operation reads the file and updates the records. Rather than performing the entire operation in one go, mini-batch scans of 1000 records are performed. To measure the efficiency of the sort operation, a file containing 1 million, 100 byte records is used.

Elapsed time and time-weighted cost of each operation is measured. The five year cost to run the transaction processing system is measured. Finally, performance/cost is calculated to compare with other transaction processing systems.

**Benchmarking Simple Database Operations (1987)**

Rubenstein et al. [21] points out that most of the engineering applications such as Computer Aided Design (CAD), Computer Aided Manufacturing (CAM), and Computer Aided Software Engineering (CASE) issue simple queries on a single
database. For such applications the response time should be in the order of few milliseconds.

They defined a set of simple operation such as name lookup, range lookup, group lookup, reference lookup, record insert, sequential scan, and database open. These simple operations were performed on INGRES, UNIFY and RAD-UNIFY database management systems.

A simple schema consisting of person, author and document was defined to run simple operations. Rubenstein et al. identified that the performance of a database system can be increased significantly by applying different strategies such as caching data in memory, avoiding query optimization, and by using a database that resides on workstation.

**Transaction Processing Performance Council (1988)**

In 1988, the Transaction Processing Performance Council (TPC) [14] was established to measure the transaction processing capability of different database systems. The TPC metrics comprised of peak performance and price/performance. TPC benchmarks became a way to compare different database products. This forced vendors to increase the performance of their database system. Initially, database systems used to achieve 25 transactions per second when run against TPC-A benchmark. After four years many database systems improved their data processing capabilities to support more than 1000 transactions per second.

In 2009, Michael Stonebraker [26] commented that TPC benchmarks are expensive to run which prevents small vendors to participate and some of them are not being used in industry.


DeWitt [11] extended the wisconsin benchmark to evaluate the performance of parallel database systems. The metrics used to measure parallel database systems
are speedup, sizeup and scaleup. Speedup measures the effect of adding processors and disks on response time. There are two variants to measure Scaleup: (1) workload is increased by adding the proportional amount of disks and processors (2) workload is increased but system configuration is kept constant. The second variant is also known as Sizeup. He performed selections and join queries on a Gamma database machine. In his experiment, relations containing 0.1 million, 1 million and 10 million tuples were declustered among 30 processors with disks.

2.2.2 Benchmarking NoSQL database

Yahoo Cloud Serving Benchmark (2010)

Yahoo Cloud Serving Benchmark [10] is the first noteworthy attempt to benchmark new generation of cloud serving systems. Different database systems are optimized for different applications. To achieve this, they make different trade-offs such as Read Performance vs. Write Performance, Latency versus Durability, Synchronous versus Asynchronous replication and Row-based or Column-based data partitioning. Five type of workloads were defined in YCSB framework to reveal the trade-off decisions made by the database systems.

YCSB has two benchmarking tiers: performance and scaling. A system has good performance if it can provide the desired latency with the fewer servers. Performance is similar to the sizeup metric of the wisconsin benchmark. Scaling tier measures the performance of the system as machines are added. This tier has two metrics: scaleup and elastic speedup which are equivalent to scaleup and speedup metrics of the Wisconsin benchmark.

Cooper et al. ran core workloads of YCSB against Cassandra, sharded MySQL, PNUTS and HBase and identified underlying trade-off decisions. For example, Cassandra and HBase have lower latency for update operation but higher latency for read operations than PNUTS and MySQL. Identifying such patterns helps a user to select a database system that fits their application.
YCSB++ (2011)

As YCSB only performs simple operations on a single database sever, Patil et al. [19] developed YCSB++\(^1\) to perform distributed testing by extending the YCSB framework. YCSB++ uses Apache ZooKeeper\(^2\) to coordinate with different nodes. YCSB++ can also perform tests to measure the following:

1. Consistency: Many data stores relax consistency in order to provide high availability. YCSB++ can measure the consistency of a data store by using Apache Zookeeper’s producer-consumer abstraction. Clients that perform insert or update operations act as producers. Consumer clients remove a key from the queue and read it from table store. If the read operations serves stale data then the consumer puts it back in the queue.

2. Table pre-spliting: Data stores like HBase and Accumulo index the table in a B-tree. The nodes of the B-tree are referred to as tablets. A tablet is represented as one or more files in the Hadoop file system which contains rows in the key space. All nodes in the cluster act as tablet servers. When a tablet overflows, the tablet sever splits the tablet, creates a new tablet on another server and transfers the rows on to the new tablet. This split operation affects the performance which can be avoided by pre-spliting the table using a priori knowledge. YCSB++ can pre-split the table based on specified split points.

3. Bulk upload: Many data stores support bulk operations. YCSB++ can interface with a specialized bulk load mechanism which can process the incoming request and store the data in a properly formatted file. YCSB++ uses an import API to communicate with the data store to perform bulk operations.

4. Server Side Filtering: Server side filtering reduces the client’s computation and the traffic between server and client. There are 4 server side filters defined in YCSB++. A filter on row keys and row values returns entire row if

---

1[http://www.pdl.cmu.edu/ycsb++/]
2[https://zookeeper.apache.org/]

it matches the specified pattern. Another type of filters returns the column name and the cell value, if the column name or column value matches the specified pattern.

5. Access Control: YCSB++ can test for schema and cell level access control. Schema level access control is applicable to the entire table and the columns within a table whereas cell level access control has checks for a cell. To perform access control test, YCSB++ needs access control lists and credentials for each operation.

YCSB++ uses Otus to collect data from different nodes and store it on a central repository which can be used to visualize the benchmark’s results.

**Big Data benchmark (2013)**

Big Data Benchmarking [3] is a community activity initiated by the Center for Large-scale Data Systems Research. They defined an end-to-end application layer benchmark which is applicable to big data applications. They identified 4 important characteristics of a Big data benchmark.

- Simplicity - A benchmark should be simple and focused to identify the relevant aspects.
- Ease of benchmarking - Cost of implementation and executions should be justifiable.
- Time to market - A benchmark should be updated to reflect the latest technology advancements.
- Verifiability of results - Results of benchmarks should be verifiable and the process to verify results should be low.

To achieve better results, database vendors may build larger systems and execute the benchmark on them. To prevent such a scenario, the Big Data Benchmarking
Community defined a reward based system where the efficiency of the system is measured.

Choosing workloads to perform big data benchmarks is a hard task. A single workload cannot represent the different use cases of big data applications, hence the community decided to define workloads using a pipeline based approach. Additionally, an extension to the TPC benchmark was also proposed to define the workload. There are 4 steps to execute the big data benchmark which are system setup, data generation, data load and execute application workload.

**Benchmarking Eventual Consistency (2014)**

Bermbach et al. [4] performed experiments on Amazon S3 to measure the staleness, and Monotonic Reads Consistency. They performed 8 different experiments between 2011 and 2013. To perform experiments, 12 reader instances and a write instance were deployed on Amazon EC2. The writer instance writes data to the data store and the reader instances poll the data. The staleness was measured by calculating the time reader takes to identify the change in the data.

During 2011 and 2013, staleness was higher which means that a web shop application deployed on Amazon S3 would oversell the products. On the other hand, during 2012, staleness results improved which would have decreased the probability of overselling. It is very important for such application to find underlying patterns to make sound decision. For example, web shop can over provision the products to compensate overselling. The annual cost of running such benchmarks to measure staleness was slightly lower than 20,000 USD.

As it incurs high cost to perform such experiments, Bermbach et al. developed an economical solution called indirect consistency monitoring. Rather than monitoring staleness, it monitors Key Performance Indicators (KPI) which are directly affected by consistency. For example, staleness of the web shop application can be measured by keeping track of the number of times the web shop oversells the product. There are four steps to set this tool up: identify datastore interaction patterns, identify
potential conflicts between patterns, identify affected KPIs, and identify and Track suitable KPI.

**Sickstore (2015)**

Single-node Inconsistent Key-value Store (Sickstore) was developed to validate the results of benchmarking tools. It simulates distributed key-value database by using an internal backend, virtual servers, and a processing layer. The processing layer consists of staleness generator and query handler. Sickstore can generate precise amounts of a staleness window per server.

Wingerath et al. configured sickstore to demonstrate 1000ms of staleness and measured the staleness using YCSB++. For the sake of simplicity, single thread was used on reader and writer instance. The staleness was measured for 200 requests. YCSB++ reported staleness of 1900ms for more than half of the requests. Using sickstore, Wingerath et al. concluded that the staleness measured by YCSB++ is inaccurate.

### 2.3 Tools used for Benchmarking

**Database Benchmark**

Database Benchmark is a .NET based open-source tool to stress-test the database system. Database Benchmark measures insert speed, read speed and size of database after performing read and insert operation. It has an advanced data generator which generates real life data streaming.

The latest version of Database Benchmark, v3.0.0-final, was released in June 2015. The most recent code was commit was on June 16, 2016.

3 https://github.com/STSSoft/DatabaseBenchmark
HammerDB\textsuperscript{4}

Hammerora was specifically designed to benchmark Oracle database systems. Later on, additional support for different database systems was included. Hence, the name was changed to HammerDB.

HammerDB is a multi-threaded benchmarking tool which supports dynamic scripting. It has an AutoPilot feature which helps to run a test multiple times by different virtual users. There is inbuilt support for TPC-\textsuperscript{5}C and TPC-\textsuperscript{6}H workload in HammerDB. It can be used to benchmark Oracle, SQL Server, Trafodion SQL on Hadoop and so on.

HammerDB-2.22 was released on January 27, 2017. Repository commits are current.

JMeter\textsuperscript{7}

Apache JMeter is a Java based application which can simulate a heavy load on the target machine. It can be used to load-test different protocols, servers and applications. Only JDBC supported databases can be benchmarked using Apache JMeter. Results of test can be captured in various formats such as summary report, aggregate graph, results in table and so on. One can either use terminal or user interface to perform simple and distributed load testing.

Apache Jmeter 3.1 was released on November 19, 2016.

\textsuperscript{4}http://www.hammerdb.com/
\textsuperscript{5}http://www.tpc.org/tpcc/
\textsuperscript{6}http://www.tpc.org/tpch/
\textsuperscript{7}http://jmeter.apache.org/usermanual/build-db-test-plan.html
Real Application Testing

Real Application Testing is owned by Oracle. It helps to reduce system instability issues. It features SQL Performance Analyzer (SPA), database replay, custom workload creation and so on. The SPA feature keeps track of any changes to the database system and assesses the impact of changes in a SQL execution plan. By using the database replay feature one can capture current workload and replay it later.

Real Application Testing is propriety software.

SwingBench

Swingbench is an open source tool written in Java to benchmark Oracle databases. It provides six benchmarks by default which are order-entry, sales-history, calling-circle, stress-test, json and TPC-DS like benchmark. One can also define a custom benchmark by modifying the stored procedures. It measures Response time, Transactions per second, CPU and disk IO.

The latest version of SwingBench is 2.5. It was released on March 15, 2016.

Tsung

Tsung is distributed load testing tool developed in Erlang language. It supports different protocols and can be used to test HTTP, MySQL, PostgreSQL, LDAP, and Jabber/XMPP servers. It generates realistic traffic by randomizing the user think-times and the arrival rate. There are only four types of requests available for MySQL and PostgreSQL which are connect, authenticate, execute and close.

Tsung 1.6.0 was released on July 20, 2015.

---

8https://www.oracle.com/database/real-application-testing/index.html
9http://dominicgiles.com/swingbench.html
10http://tsung.erlang-projects.org/
Benchmark Factory

Benchmark Factory is the proprietary product of Quest Software. It has in-built support for TPC-C, TPC-H, TPC-D, TPC-E benchmarks. It can simulate users and transactions which can be used to identify performance problems. This prepares the database administrator to develop a strategy to fix it.

Benchmark Factory can capture the existing workload which can be replayed against a different database system to compare performance. Apart from performing a benchmark, it can be used to determine the correct number of Oracle Real Application Cluster nodes to minimize the costs and assist the database administrator to migrate from the older version to the latest version of the database.

Yahoo Cloud Serving Benchmark

Yahoo Cloud Serving Benchmark (YCSB) is an open source benchmarking tool created by Yahoo. There are five core workloads defined in YCSB namely Update heavy, Read heavy, Read Only, Read latest and Short ranges which helps to reveal trade-off decision made by different database systems. YCSB has four inbuilt distribution algorithms to create Uniform, Zipfian, Latest and Multinominal distribution to make random choices about when to generate load, which record to read or write and so on. It has an extensible architecture which allows YCSB to adapt to a new database system. A custom workload can also be defined by specifying the mixture of read, write, update and scan operation.

YCSB 0.12.0 was released on December 12, 2016.

11https://www.quest.com/products/benchmark-factory/
12https://github.com/brianfrankcooper/YCSB
OLTP Bench\textsuperscript{13} is an open source software written in Java. It features 15 workloads categorized into 3 groups namely: Transactional, Web-oriented workload and Feature testing. Transactional workload includes seven different types of workloads which are write-intensive. The web-oriented workload is designed to imitate a real application. Feature Testing workload consists of light-weight micro-benchmarks. This workload tests different features of the database.

It supports three different models to invoke transactions: (1) Closed loop in which a fixed number of worker threads are initialized with a random think time between requests. (2) Open loop - Requests are invoked randomly. (3) Semi-Open - It is an extension of Open loop approach. Here, workers pause for a random amount of time before submitting a new request.

OLTP-Bench can be deployed on multiple machines to efficiently benchmark distributed database systems.

\subsection*{2.4 Virtualization Environment}

Hypervisors play a key role in virtualization. It creates and runs virtual machines. A hypervisor can be classified into two categories: a type 1 hypervisor directly runs on the machine whereas a type 2 hypervisor runs on top of the operating system.

\textbf{Kernel-based VM (KVM)}\textsuperscript{14}

KVM \textsuperscript{9} is an open-source type-2 hypervisor provided by Red Hat. It is shipped with the Linux Kernel since the release of version 2.6.20. KVM needs qemu, libvirt and bridge-utils which provide guest emulation, management, and bridge networking respectively. It treats a VM as a process and virtual CPUs as threads within a

\textsuperscript{13}https://github.com/oltpbenchmark/oltpbench
\textsuperscript{14}https://www.linux-kvm.org/page/Main_Page
process. It performs memory and I/O management and allows a VM to perform an operation on a physical processor. It supports asset management, snapshot creation, VM migration and live migration.

KVM 1.2.0 was released in September 2012. KVM is commonly used in OpenStack.

**Hyper-V**

Hyper-V is a type-1 hypervisor from Microsoft. It consists of one parent partition and multiple child partitions. The parent partition uses drivers to access hardware and works as a broker between the hardware and child partitions. Guest operating systems are hosted on child partitions. Some of the features of hyper-v are host resource protection, nested virtualization, performance reports.

**VMware ESX/ESXi**

VMware ESX/ESXi is type-1 hypervisor provided by VMware which is built on x86 hardware. It is a thin software layer of 150MB. VMKernel, device drivers, file systems, processes such as Virtual Machine Monitor (VMM), Virtual Machine Executable (VMX) process and Common Information Model are principal components of the ESXi hypervisor. It supports live migration, dynamic resource allocation and virtual firewalls.

The latest version of VMWare ESX/ESXi is 6.5 which was released on 15 November 2016.

**Xen**

Xen is a type-1 hypervisor which runs on 32-bit and 64-bit processors. In Xen, virtual machines are referred to as Domains. Domain0 and DomainU are two

---

16 http://www.xenproject.org/
type of domains. Dom0 has direct access to hardware. Xen hypervisor performs memory partitioning and CPU scheduling. DomU uses Xen hypervisor as a medium to communicate with dom0 to use resources. Amazon EC2, Rackspace Cloud are some of the cloud service providers who use Xen hypervisor.

**VirtualBox**[^17]

Virtualbox is an open source type 2 hypervisor owned by Oracle. It allows the host to run an actual operating system, but it consumes substantial resources. It is widely used for testing an application on a different environment. It provides features such as VM cloning, asset management, shared resource pools.

VirtualBox v5.1.18 is the current stable version released in March 2017.

**Parallels Desktop**[^18]

Parallels Desktop is a type-2 hypervisor. It maps the host’s resources to VM’s resources. It supports different operating systems on the host machine and is widely used to run Windows on Mac machines. It can be used to test web applications among different environments.

The latest version of Parallels Desktop is 12.1.3 which released in February 2017.

[^17]: https://www.virtualbox.org/
[^18]: http://www.parallels.com/products/desktop/
Chapter 3

Proposed Approach

In this section we look into the parameters that affect the results of benchmarking and discuss our approach to design the experiments.

3.1 Our Approach

To perform a successful benchmark, one needs to carefully create experiments which can provide meaningful results. The steps we followed to identify the effects of virtualization are showed in Figure 3.1.

![Figure 3.1: Our approach to design the experiments.](image-url)
We will perform initial experiments on a single virtual machine (VM). Then we use the results of this experiments to create experiment on multiple VM.

On a single VM, we will run stress test to identify how time affects the results of benchmark. Then we change the thread configuration to run stress tests and identify the thread configuration which takes less amount of time to complete tests. Using the results, we will benchmark micro and small VM. For each thread configuration, we run stress tests five times. Then we performed T-test to identify the consistency in VM’s performance. Identifying VM configuration which performs consistently with different thread configuration allows us to compare the performance of VMs.

After identifying the VM configuration which provides consistent results, we use it to identify how different VM responds to stress tests ran in sequential and parallel manner.

3.2 Parameters that affect Benchmarking

Seltzer [23] points out how benchmarking misleads the industry. Rather than relying on the published benchmarks, he recommends performing a benchmark as according to the application’s need.

Modern applications follow n-tier architecture. Moreover, modular architecture allows flexibility. Figure 3.2 represents 3 tier architecture which includes:

- Presentation Tier: Presentation tier helps users to communicate with the application via the user interface. For a typical web application, the browser is presentation tier.

- Logic Tier: This tier consists of actual logical code that process user’s request and perform different operations. It can be an application deployed on a web server.
- **Data Tier**: Data tier is in charge of managing the data. It includes a mechanism to persist and access data. Logic tier communicates with data tier using API.

![3 Tier Architecture of the modern applications.](image)

**Figure 3.2**: 3 Tier Architecture of the modern applications.

Users of the application are on a different machine than the server. We decided to place benchmarking client and server on the same network to exclude the effect of networking. Parameters that affect benchmarking in such environment are:

- **Server Configuration**: Current database systems to take maximum advantage of the available resources. If there are multiple processors available, then database systems automatically occupy them to scale up according to their need.

- **Database**: The decision about what type of database to choose depends on the type of application. Once this is identified; different database systems are reviewed to find a sound solution. Different database systems employ various strategies to retrieve, manage and remove data which creates a difference in their performance.

- **Database Configuration**: As data grows, different issues starts to emerge. One needs to fine tune database systems at particular intervals to get maximum performance. There are many tools which assist users to do so [29][18][17]. Many cloud providers offer database-as-a-service in which they would take care of administrative tasks such as installation, upgrade, and backup of the database, synchronization among replicas.

- **Configuration of the benchmarking tool**: Different applications exhibit a different type of workload. Hence, benchmarking tools needs to be configured properly to generate application specific workload.
To prove the hypothesis that virtualization affects the database benchmarking, we performed experiments by changing VM size and thread configuration.
Chapter 4

Experimental Design and Implementation

Building from the information discussed in the prior chapter we have designed and implemented an experimentation environment using a emulation environment called VINE (discussed later). Figure 4.1 shows the high-level design of our experiment environment. We used following tools to conduct our experiments:

4.1 Yahoo Cloud Serving Benchmark (YCSB)

YCSB tool can be used to benchmark both SQL and NoSQL database systems. One needs to configure a YCSB tool to provide schema specific details on how to perform the benchmark. The custom workload can be easily defined by specifying a mixture of operations. There are options to specify the number of operations to perform, maximum execution time and so on.
Figure 4.1: A high level design diagram of the environment used to perform experiments.

4.2 Implementation of Java program

We run an experiment in three phases. In phase 1, we load the database by generating requests using YCSB framework. Once the experiment is completed, we reset the database. Later on, we collect and process the result of the entire experiment. We wrote a Java program that extends the YCSB framework to vary client thread configuration, reset the database, and aggregate the results of the experiment. The output of the Java program is a comma separated values (CSV) file which includes thread configuration, elapsed time, throughput, and latency. After the end of each experiment, our Java program stops execution for 10 seconds to ensure that the earlier experiment does not influence the next experiment.

4.3 MySQL database

MySQL database\textsuperscript{1} is one of the widely used relational database systems used in industry. Commercial and community versions of MySQL are available to use. We used MySQL 5.5.54 community edition for our experiments.

For our experiments, we used the following schema:

\textsuperscript{1}https://www.mysql.com/
CREATE TABLE usertable (  
    YCSB_KEY VARCHAR(255) PRIMARY KEY,  
    FIELD0 TEXT, FIELD1 TEXT,  
    FIELD2 TEXT, FIELD3 TEXT,  
    FIELD4 TEXT, FIELD5 TEXT,  
    FIELD6 TEXT, FIELD7 TEXT,  
    FIELD8 TEXT, FIELD9 TEXT  
);  

As data grows the performance of the database decreases. Hence, we reset the database before we run experiments. There are three strategies to reset a database: delete all records, truncate Table, drop Table. Approach 1 needs to be executed in 2 steps. In the first step, a delete statement is executed and in the second step, we need to reset the index. This approach is time-consuming and generates an additional log. Truncate table deletes all the records in a table and resets the index. If a table is dropped, then one needs to recreate all the constraints, indexes and foreign keys. Hence, we reset the relation using the truncate table command.

4.4 Tableau

To analyze and visualize the results of our experiments we used Tableau\(^2\) Tracking the results of the experiment online introduces additional processes on the CPU. To reduce this overhead, we analyze results of experiments once experiments are completed.

4.5 Virtual Infrastructure Network Emulator (VINE)

VINE \(^{28}\) is a virtualization environment that helps to create different network topologies. It offers a variety of configurations to build virtual machine according

\(^{2}\)https://www.tableau.com/
to one’s preference. One can easily access the provisioned virtual machine over SSH and modify it to run different services such as FTP, SMTP, MySQL and so on. VINE can adapt different virtualization platform such as Virtualbox, OpenStack and so on and provides a consistent view to end user.

To make sure that the entire experiment has identical images; we created two virtual machines and installed the MySQL database and YCSB tool on it. VINE offers a feature to snapshot VMs. We created snapshots for both VM and used them later on to perform experiments. To exclude the effect of networking, we created an additional interface on each VM and created a private network in which server and client are connected directly. Figure 4.2 shows a sample testbed which includes 7 YCSB clients directly connected with 7 MySQL machines in the internal network.

Combined, all these tools and frameworks were used to run our experimentation. The next chapter provides a summary of our experimental results, followed by our analysis.
Figure 4.2: A sample testbed which has 7 MySQL servers connected with 7 YCSB clients.
Chapter 5

Experiments and Analysis

In this chapter, we will look into the details of the experiments we have performed to identify the effects of virtualization in benchmarking.

VINE offers many default configuration options, but for our experiments, we used the following configuration.

- **Micro Image** has 1 CPU with 256 MB RAM and 3 GB of disk space.
- **Small Image** has 1 CPU with 2048 MB RAM and 20 GB of disk space.

We created an additional interface on both the YCSB client and the MySQL server. The YCSB client uses this interface to send requests to insert data into the MySQL server. This allows us to fully utilize the network to benchmark the database system.

**Overview of the experiment**

Figure 5.1 shows the design of the experiment. The experiment consists of stress tests designed to insert a specified number of records into the MySQL database. We choose the number of threads each stress test should be executed with. For example, if we run a stress test to insert 100,000 records with 5 threads, 5 connections would be opened and 100,000 records would be inserted.
5.1 Identify the effect of time

To identify how time affects the results of the benchmark, we benchmark MySQL database running on a micro image. The stress test inserts 1 million records in MySQL using 25 threads.

Figure 5.2 depicts the throughput vs time and latency vs the time diagram. As time passes, data grows and buffer memory starts to fill up resulting in reduced throughput. This also increases the response time. There is an ‘L-shaped’ pattern in throughput. MySQL stores data in B+tree data structure [22]. When data overflows at the root node, two new nodes are created and data is split among newly created nodes. Because of split operation, we see an ‘L-shaped’ throughput pattern.
5.2 Identify the effect of running stress tests with different number of threads

To benchmark database systems, one needs to choose the right number of threads to run the stress tests. If we use a lower number of threads then the database systems are underutilized whereas if a higher number of threads are used then the database systems are overutilized. To identify the optimal number of thread configuration, we ran stress tests to insert 100,000 records on a MySQL database by varying the thread count configuration. Each stress test was executed five times. The experiment stops when the database service halts on MySQL server.

Figure 5.3 is a box-plot diagram which shows the result of stress tests. While running stress tests with 65 threads, buffers overflowed which triggered database shutdown. As we increase the number of threads, we see the total time to complete stress tests reduces. But if we run experiments with more than 42 threads, the total time to complete the stress test increases. It takes the least amount of time to complete stress tests while running tests with a number of threads between 16
and 35. From this range, we chose 20, 25, 30 and 35 to run our next experiments. These points represent the entire range and they are evenly distributed.

![Figure 5.3](image)

**Figure 5.3:** Time to complete stress test to insert 100,000 records for each thread configuration.

### 5.3 Comparing the effects of multiple threads

In this experiment, we benchmarked MySQL database running on the micro image by inserting 1 million records using 20, 25, 30 and 35 threads. Figure 5.4 shows the throughput and latency pattern for 20, 25, 30 and 35 threads. As we increase the number of threads latency increases whereas throughput overlaps.

To identify the difference in the throughput pattern, we performed curve fitting on the logarithmic equation

\[ y = a \times \log(x) + b \]

and then T-test was run with the null hypothesis that the log coefficient of the equation is equal. Figure 5.5 shows throughput pattern for 20, 25, 30 and 35
threads. The throughput for each thread is represented in Table 5.1. The result of the T-test is shown in Table 5.2, indicating micro images exhibit different throughput pattern for various thread configurations. Therefore, micro images do not perform consistently.

**Figure 5.4:** Average throughput vs. time and average latency vs. time result colored by the thread configuration.

**Figure 5.5:** Throughput vs. time for 20, 25, 30 and 35 threads colored by individual runs.
Table 5.1: Equation to represent the throughput pattern of different thread configuration on micro image.

<table>
<thead>
<tr>
<th>Thread Configuration</th>
<th>Throughput equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>( y = -689 \times \log(x) + 5047 )</td>
</tr>
<tr>
<td>25</td>
<td>( y = -691 \times \log(x) + 5061 )</td>
</tr>
<tr>
<td>30</td>
<td>( y = -668 \times \log(x) + 4917 )</td>
</tr>
<tr>
<td>35</td>
<td>( y = -635 \times \log(x) + 4712 )</td>
</tr>
</tbody>
</table>

Table 5.2: The result of T-tests for the experiment performed on the micro image.

<table>
<thead>
<tr>
<th>Thread Configuration</th>
<th>25</th>
<th>30</th>
<th>35</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Accepted</td>
<td>Accepted</td>
<td>Rejected</td>
</tr>
<tr>
<td>25</td>
<td>Rejected</td>
<td>Rejected</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td>Rejected</td>
<td></td>
</tr>
</tbody>
</table>

5.4 The effects of multiple threads for a different VM configuration

In this experiment, we changed the VM configuration to a small image (1CPU/2048MB/30GB). MySQL database was benchmarked to insert 1 million records using 20, 25, 30 and 35 threads. Each stress test was run for five times.

Figure 5.6: Throughput vs. time for 20, 25, 30 and 35 threads colored by individual runs performed on MySQL database installed on small image.

Figure 5.6 shows the throughput pattern for 20, 25, 30 and 35 threads. To identify the difference in throughput pattern, we performed curve fitting and then run T-test
with the null hypothesis stating that the log coefficient of the equations is equal. Table 5.3 shows the throughput equation for each thread configuration. The result of T-tests are shown in Table 5.4, indicate that small images perform consistently. Thereby we used small image configuration for our next experiments.

**Table 5.3**: Equation representing the throughput pattern of different thread configuration on small image.

<table>
<thead>
<tr>
<th>Thread Configuration</th>
<th>Throughput equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>$y = -510 \times \log(x) + 5320$</td>
</tr>
<tr>
<td>25</td>
<td>$y = -400 \times \log(x) + 4847$</td>
</tr>
<tr>
<td>30</td>
<td>$y = -395 \times \log(x) + 4829$</td>
</tr>
<tr>
<td>35</td>
<td>$y = -389 \times \log(x) + 4784$</td>
</tr>
</tbody>
</table>

**Table 5.4**: The result of T-tests for the experiment performed on the small image.

<table>
<thead>
<tr>
<th>Thread Configuration</th>
<th>25</th>
<th>30</th>
<th>35</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Accepted</td>
<td>Accepted</td>
<td>Accepted</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td>Accepted</td>
<td>Accepted</td>
</tr>
</tbody>
</table>
| 30                   |         |         | Accepted |}

#### 5.5 Running tests on multiple VMs in sequential manner

For this experiment, we took 5 small VMs running MySQL service. These VMs were benchmarked by inserting 1 million records using 20, 25, 30 and 35 threads. Each stress test was run 5 times.

Figure 5.7 shows the throughput pattern for 20, 25, 30 and 35 thread configuration. MySQL1, MySQL2, and MySQL5 take about 400 seconds to complete the stress tests whereas MySQL3 and MySQL4 take about 600 seconds. We applied curve fitting for the VMs and the throughput pattern of each VM is shown in Table 5.5. To understand the difference in throughput pattern, we plotted a graph.
Figure 5.7: Throughput vs. time for 20, 25, 30 and 35 threads colored by individual runs performed on MySQL database installed on small image.

for each equation as shown in Figure 5.8. The upper region of the figure shows the graph for MySQL1, MySQL2, and MySQL5 whereas the lower region represents the throughput equations for MySQL3 and MySQL4.

**Table 5.5**: Equation to represent the throughput pattern of different VM.

<table>
<thead>
<tr>
<th>VM</th>
<th>Throughput equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM1</td>
<td>$y = -396 \times \log(x) + 4808$</td>
</tr>
<tr>
<td>VM2</td>
<td>$y = -427 \times \log(x) + 5007$</td>
</tr>
<tr>
<td>VM3</td>
<td>$y = -243 \times \log(x) + 2954$</td>
</tr>
<tr>
<td>VM4</td>
<td>$y = -247 \times \log(x) + 3016$</td>
</tr>
<tr>
<td>VM5</td>
<td>$y = -425 \times \log(x) + 4950$</td>
</tr>
</tbody>
</table>
The result of the experiments show that identical VMs may respond differently to the same stress test. On further investigation, we found that the VMs created on VINE are provisioned on one of three physical servers. Figure 5.9 shows the placement of those VMs. Comparatively, the VMs provisioned on compute node 3 took higher amount of time to complete stress tests than the VMs created on compute node 1 and 2.

MySQL1 and MySQL2 are on same compute node but their logarithmic coefficients are different. It is possible that the difference in the coefficient might be attributed to the placement of benchmarking client. In this case, MySQL1 was benchmarked using YCSB1 which is on a different compute node. Hence, network overhead might have affected the result of MySQL1. To verify whether network overhead influences the results, we performed a T-test. The null hypothesis is that the logarithmic coefficients are equal. The result of the T-test is shown in Table 5.6. There is no difference in the throughput pattern of MySQL1 and MySQL2. This means that the network overhead does not influence the result of the benchmark in our environment.
Figure 5.9: The placement of VM on physical server.

Table 5.6: The result of T-tests for the experiment performed with 5 small VMs running the MySQL service.

<table>
<thead>
<tr>
<th>Thread Configuration</th>
<th>MySQL2</th>
<th>MySQL3</th>
<th>MySQL4</th>
<th>MySQL5</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL1</td>
<td>Accepted</td>
<td>Rejected</td>
<td>Rejected</td>
<td>Accepted</td>
</tr>
<tr>
<td>MySQL2</td>
<td>Rejected</td>
<td>Rejected</td>
<td>Accepted</td>
<td></td>
</tr>
<tr>
<td>MySQL3</td>
<td>Accepted</td>
<td>Rejected</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MySQL4</td>
<td>Accepted</td>
<td>Rejected</td>
<td></td>
<td>Rejected</td>
</tr>
</tbody>
</table>
5.6 Running tests on multiple VMs in a parallel manner

To identify how VMs behave when stress tests are run in a parallel manner we created 2 VMs per physical host. We run stress test by inserting 1 million records using 20, 25, 30 and 35 threads. Each stress test was run five times.

Figure 5.10 shows the throughput pattern for different VMs. We do not see a consistent pattern in throughput. VINE uses KVM to provide virtualization. In KVM, each VM is a process and virtual CPU is a thread within the process [9]. Therefore, the variation in throughput pattern may be attributed to the thread scheduler used in KVM.

Figure 5.10: Running stress tests in parallel.
The results of our experiments indicated that in our virtual environment, VMs may exhibit different behavior. Often users do not have control over the virtual environment and do not know the details of the virtualization platform. To benchmark database system in such an environment, one should not rely on the results of the experiments performed on a single VM.

Additionally, while performing experiments using multiple VMs; one should spread the stress tests between different VMs. By doing so, the effects of VMs running on the slower physical host are minimized.
Chapter 6

Recommendations

When we run a resource intensive task in virtual machine, the hypervisor has to allocate the resources to virtual machines. This can be used to measure the resource allocation capabilities of the hypervisor. We can change the virtualization platform to VMWare or Xen and run multiple stress tests on different VM in parallel. This will allow us to compare the efficiency of different hypervisors.

Many cloud vendors offer database-as-a-service which automatically installs and patches database systems. It can be configured to take automated backup and synchronizes replicas. It takes care of mundane tasks for database administrators and allows them to spend more time on designing and fine tuning the database system[24]. A possible direction of future research includes comparing the performance of databases installed on a VM with the database-as-a-service offering.
Chapter 7

Conclusion

In this thesis, we performed benchmarks in a virtual environment to identify the effects of virtualization. We looked into different types of database, their design principals, different approaches to benchmark database systems, and tools to benchmark them. To identify the effects of virtualization, we ran five different experiments by changing the number of VMs, VM configuration and thread configuration.

Rather than using throughput discrete numbers for comparison, we defined throughput as a function of time which allows us to compare VMs mathematically. Our experiments show that the performance of a VM is highly influenced by the physical host on which the VM is created.
Bibliography


