BENCHMARKING POPULAR BACKEND FRAMEWORK

VIKROM NARULA

A SENIOR PROJECT SUBMITTED IN
PARTIAL FULFILMENT
OF THE REQUIREMENTS FOR
THE DEGREE OF BACHELOR OF SCIENCE
(COMPUTER SCIENCE)
MAHIDOL UNIVERSITY INTERNATIONAL COLLEGE
MAHIDOL UNIVERSITY
2022

Senior Project

entitled

BENCHMARKING POPULAR BACKEND FRAMEWORK

was submitted to the Mahidol University International College, Mahidol University
for the degree of Bachelor of Science (Computer Science)
on
18th July 2022

.....
Mr.Vikrom Narula
Candidate

.....
Dr. Rachata Ausavarungnirun
Advisor

.....
Asst. Prof. Dr. Aram Tangboonduangjit
Chair of Science Division
Mahidol University International College
Mahidol University

.....
Dr. Boonyanit Mathayomchan
Program Director
Bachelor of Science in Computer Science
Mahidol University International College
Mahidol University

ACKNOWLEDGEMENTS

I would like to express gratitude to our project advisor, Dr. Rachata Ausavarungnirun who provided me with valuable guidance and encouragement throughout this project. Additionally, I would like to thank our advisor Dr.Kritya Bunchongchit for providing endless support throughout college life as well as Asst. Prof. Dr. Kanat Tangwongsan, Dr. Sunsern Cheamanunkul and Dr. Weerapong Phadungsukanan for providing me knowledge and experiences in this field of study and in life for the past years. Finally, I would like to thank all of our friends and peers for their mental support, motivation, and guidance throughout the bachelor degree.

Vikrom Narula

BENCHMARKING POPULAR BACKEND FRAMEWORK.

VIKROM NARULA 6081050 ICCS/B

B.Sc. (COMPUTER SCIENCE)

SENIOR PROJECT ADVISORS : DR. RACHATA AUSAVARUNGNIRUN, (COMPUTER SCIENCE)

ABSTRACT

This project was created to explore the limitation of popular backend frameworks, from upcoming and industry standard languages. This project will help explore on standardization of benchmarking application programming interfaces. This project will mainly focus on how the application programming interface interaction with the database. The project will investigate how will the hardware resources will affect the database utilization of the application programming interface.

KEY WORDS : FRAMEWORK, RUST, PYTHON, JAVASCRIPT, BENCHMARK, TPC-C, KVM, POSTGRESQL

39 pages

CONTENTS

ACKNOWLEDGEMENTS **iii**

ABSTRACT (ENGLISH) **iv**

1 Introduction **1**
1.1 Motivation 1
1.2 Purpose 1

2 Tech Stack **2**
2.1 Git 2
2.2 Kernel-based Virtual Machine 2
2.3 Django 2
2.4 Express.js 3
2.5 Actix-Web 3
2.6 PostgreSQL 3
2.7 Putting Everything Together 3
2.8 Virtual Machine Setup 5

3 Methodology **6**
3.1 Metric of Interest 6
3.1.1 Scalability with increasing number of CPU threads 6
3.1.2 Impact from the size of the main memory relative to performance 6
3.1.3 Impact from the size of database to throughput and latency 6
3.2 TPC-C Emulation 7
3.2.1 New Order Transaction 7
3.2.2 Payment Transaction 8
3.2.3 Order Status Transaction 8
3.2.4 Delivery Transaction 9
3.2.5 Stock Level Transaction 9
3.3 Benchmark Criteria 9

4 Performance **11**
4.1 Difference in Number of Threads 11
4.1.1 Latency 11
4.1.2 Throughput 12
4.2 Difference in Number of Main Memory 13
4.2.1 Latency 13

4.2.2 Throughput 14
4.3 Difference in Data Amount 14
4.3.1 Latency 14
4.3.2 Throughput 15

5 Qualitative Analysis 20
5.1 Ease of Code 20
5.1.1 Django 20
5.1.2 Express.js 21
5.1.3 Actix-Web 21
5.2 Documentation 21
5.2.1 Django 22
5.2.2 Express.js 22
5.2.3 Actix-Web 22
5.3 Learning Curve 23
5.3.1 Django 23
5.3.2 Express.js 23
5.3.3 Actix-Web 24

6 Quantitative Analysis 25
6.1 Development Timeline 25
6.1.1 Django 25
6.1.2 Express.js 25
6.1.3 Actix-Web 26

7 Artifacts 27
7.1 Generate Database Virtual Machine 27
7.2 Destroy Database Virtual Machine 27
7.3 Setup Database Virtual Machine 27
7.4 Reset Database Virtual Machine 32
7.5 Generate Framework Virtual Machine 32
7.6 Destroy Framework Virtual Machine 33
7.7 Setup Framework Virtual Machine 33
7.7.1 Django 34
7.7.2 Express.js 34
7.7.3 Actix-Web 34
7.8 Data Collection 34

8 Conclusion 36
8.1 Lesson learn Future Plans 36

REFERENCES 38

BIOGRAPHY **39**

CHAPTER 1

INTRODUCTION

Before for project manager to pick a framework to build their application programming interface for their application they see if the framework fits their requirements. But currently with how most frameworks are very similar to each other it getting hard to pick which framework suites what needs. Hence this project was created to benchmark popular frameworks on its utilization of database, and how the difference in hardware will affect the performance.

1.1 Motivation

With currently landscape of backend framework. It is getting harder everyday to pick the right framework to use for the requirement of a project. This project aim is to help clarify the decision making of picking framework to build your application programming interface.

1.2 Purpose

The purpose of this project is to help developer in process of picking framework to use in their project for their application programming interface. The project will show how popular framework utilize database, and how difference on hardware will affect the performance.

CHAPTER 2

TECH STACK

This project utilizes theses following technologies.

2.1 Git

Git is a source control system in all mainstream platforms through a free software license. Git allow developer to to mange the code base both privately and remotely. It allows collaboration between developer both remotely and locally. It greatest benefits it the ability to work offline, and the simplicity of set up [9].

2.2 Kernel-based Virtual Machine

Kernel-based Virtual Machine or a KVM is a virtualization technology for family of x86 CPUs. It provides an environment for developers to test or host their application by utilizing virtualization [2].

2.3 Django

Django is a Python web framework that promotes rapid development. and clean design. The reason we chose this framework is that it a good entry web framework for developer [3]. With the popularity of Python the understanding of the language will not be an issue.

For deployment we will be using `gunicorn` due to it ability to deploy worker to handle request. We scale the amount of worker depending on this formula $w = (c * t) + 1$, where w is the amount of worker, c is the amount of CPU cores, and t is the amount of threads per CPU cores.

2.4 Express.js

Express.js is a web framework that is bases on the core `http` module, and Connect component from Node.js. Those components are *middleware* which the idea behind Express.js. This allows developer to pick any libraries they need, which give high ability to customize and it gives high flexibility [7]. This framework was chosen due to its uses of JavaScript as it programming language, its high ability to be customize, and its freedom to uses libraries.

2.5 Actix-Web

Actix Web is a web framework which focuses on actor design pattern. Actix Web is a minimal framework which is powerful, and pragmatic. Actix Web uses Rust as its programming language [6]. Rust is a very new, complied memory safe language. This framework was chosen due to its minimalism and the new trend of Rust as a programming language.

2.6 PostgreSQL

PostgreSQL is a object relational database management. PostgreSQL unlike other database it supports both relational and non-relational data types. PostgreSQL is also an open source project which allows honest access to benchmarking numbers and statistic unlike propriety software [1].

2.7 Putting Everything Together

The benchmark is set up on a Ubuntu 20.04 server with Intel Core i9-9900 CPU, 16 GB of RAM, and 250 GB of NVMe SSD. The application programming interface is set up to its own KVM, the PostgreSQL database is also isolated to its own KVM. The communication between the KVM is done by NAT networking. Lastly the client who is requesting data from the application programming interface is run on the server it self. It communicates to the application programming interface KVM with NAT networking. This ensures that the hardware used for the database, and the application programming

interface are separate. The concept is better understand with Figure 2.1. Where the arrows are the communication via NAT networking.

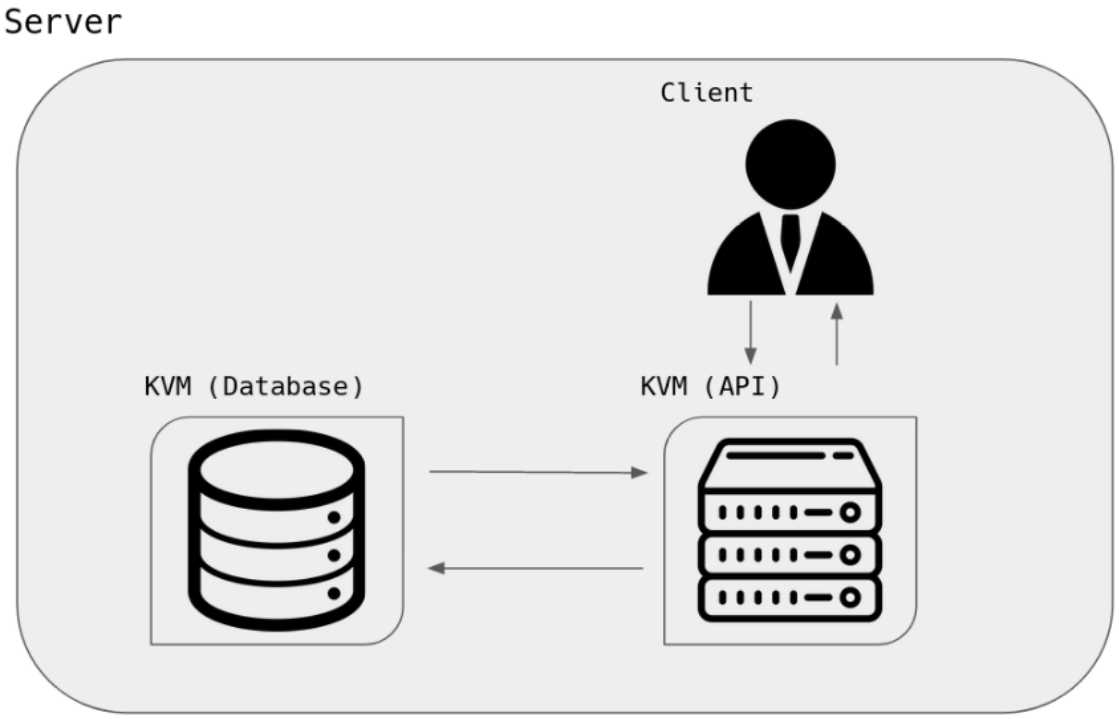

Figure 2.1: Benchmark setup

2.8 Virtual Machine Setup

For each benchmark data collection, we create a database virtual machine via KVM with the resource of 2 core of CPU, and 4 GB of RAM. We generate data with the required warehouse and migrate the data to model in the the database KVM via PostgreSQL. Afterwards we set up networking for PostgreSQL to accept connection from any IP. When virtual machine of database is up, we will generate virtual machine via KVM for the needed API. The resources of the virtual machine depends on the benchmark criteria. Afterwards we install the required software needed to run the application programming interface. Then we configure the networking of the application programming interface to connect to the database KVM. After all the set up is done we run the application programming interface. The client is run on the server to request each application programming interface endpoints to get the request time, and record it. We will use the request time data to analysis latency of the each framework.

CHAPTER 3

METHODOLOGY

The focus of the project is to see how each framework chosen will utilize hardware resources. To achieve this the framework will be benchmark on different hardware configurations.

3.1 Metric of Interest

3.1.1 Scalability with increasing number of CPU threads

By limiting the number of threads the framework has to utilize. We will be seeing how effective the framework will utilize multi-threading. We will also see will how the framework will scale with amount of threads given. For this we will be measuring the respond time in second depends on the number of threads of the CPU with 10 warehouses and 8 Gb of main memory.

3.1.2 Impact from the size of the main memory relative to performance

By limiting the size of the main memory the framework has to utilize. We will be seeing how effective the framework will utilize I/O with the main memory. We will also see will how the framework scale with amount of main memory given. For this we will be measuring the respond time in seconds depends on the number of given main memory with 10 warehouses and 4 threads of CPU.

3.1.3 Impact from the size of database to throughput and latency

By controlling the size of the database. We will be seeing how effective the framework will utilize its interact with the object-relational mapping and the database. We will see the how the framework will interact with the database. For this we will be measuring the throughput or the transaction of the database per second with 8 Gb of main memory and 4 threads of CPU.

3.2 TPC-C Emulation

TPC-C is a database model which focuses on simulation a workload of wholesale supplier company. The main workload are on multiple SQL per transaction. The hierarchy of the model are dependent the number of the warehouse. There are five transaction in TPC-C, where each transaction focuses on different workload on the database [5]. The relation between model is on the Figure 3.1.

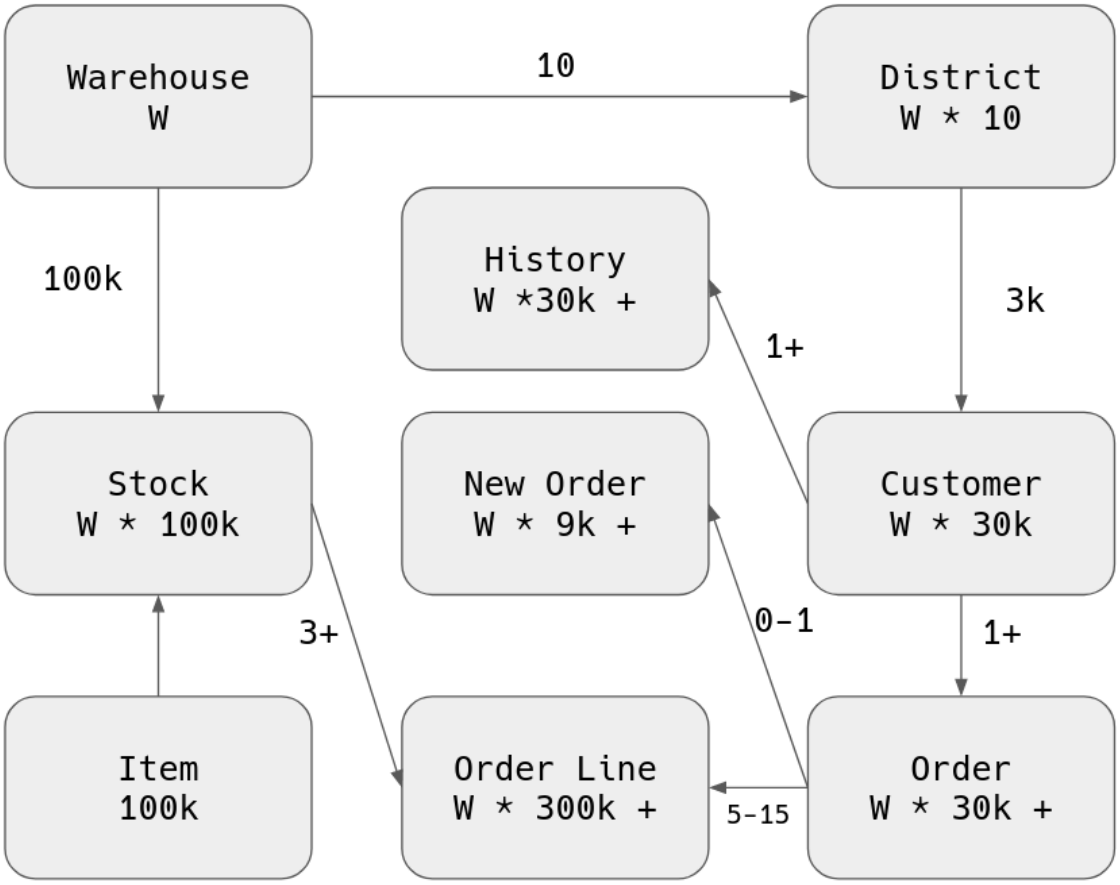

Figure 3.1: Benchmark setup

3.2.1 New Order Transaction

New Order transaction simulate customer placing on average 10 orders, and the data is update for the district, and stock level. Here is a pseudo SQL code for the New Order transaction:

```
1. Select Warehouse
```

```
2. Select District
3. Update District
4. Select Customer from the given Warehouse, and District
5. Insert Order
6. Insert New Order
7. For each Item (10 items):
    a. Select Item
    b. Select Stock
    c. Update Stock
    d. Insert Order Line
```

3.2.2 Payment Transaction

Payment transaction simulate processing of payment of the customer, updating their balance, updating their credit, and update other relational data. Here is a pseudo SQL code for the Payment transaction:

```
1. Select Warehouse
2. Select District
3. Select Customer
    a. Case 1: Select by Customer ID
    b. Case 2: Select by Customer last name
4. Update Warehouse
5. Update District
6. Update Customer
7. Insert History
```

3.2.3 Order Status Transaction

Order Status transaction simulate getting status of the order of a given customer. Here is a pseudo SQL code for the Order Status transaction:

```
1. Select Customer
    a. Case 1: Select by Customer ID
    b. Case 2: Select by Customer last name
2. Select Newest Order from Order
3. Select Order Line
```

3.2.4 Delivery Transaction

Delivery transaction simulate processing 10 pending orders, from each district in a warehouse with 10 item per order. Here is a pseudo SQL code for the Delivery transaction:

```
1. For each District in a Warehouse (10 District):
    a. Select Oldest New Order
    b. Delete New Order
    c. Select Order
    d. Update Order
    e. For each Item in Order (10 items):
        i. Select Order Line
        ii. Update Order Line
    f. Select Customer
    g. Update Customer
```

3.2.5 Stock Level Transaction

Stock Level transaction simulate getting quantity of stock for items ordered by each of the last 20 orders in a district. Here is a pseudo SQL code for the Stock Level transaction:

```
1. Select Warehouse
2. Select District
3. Count distinct Stocks from Order Line within range
```

3.3 Benchmark Criteria

The project focuses on these criteria like we have mentioned before:

1. Scalability with increasing number of CPU threads
2. Impact from the size of the main memory relative to performance
3. Impact from the size of database to throughput and latency

Hence our focus on the hardware configuration will be on CPU cores, and RAM size. With the limited resources of the server we will be able to benchmark to these criteria:

- Default benchmark of 4 CPU core with 8 GB of RAM
- Varies the CPU core from 2 till 8 cores
- Varies the RAM size from 2 GB till 8 GB
- Varies database size with warehouse amount from 1 warehouse till 25 warehouse

CHAPTER 4

PERFORMANCE

4.1 Difference in Number of Threads

4.1.1 Latency

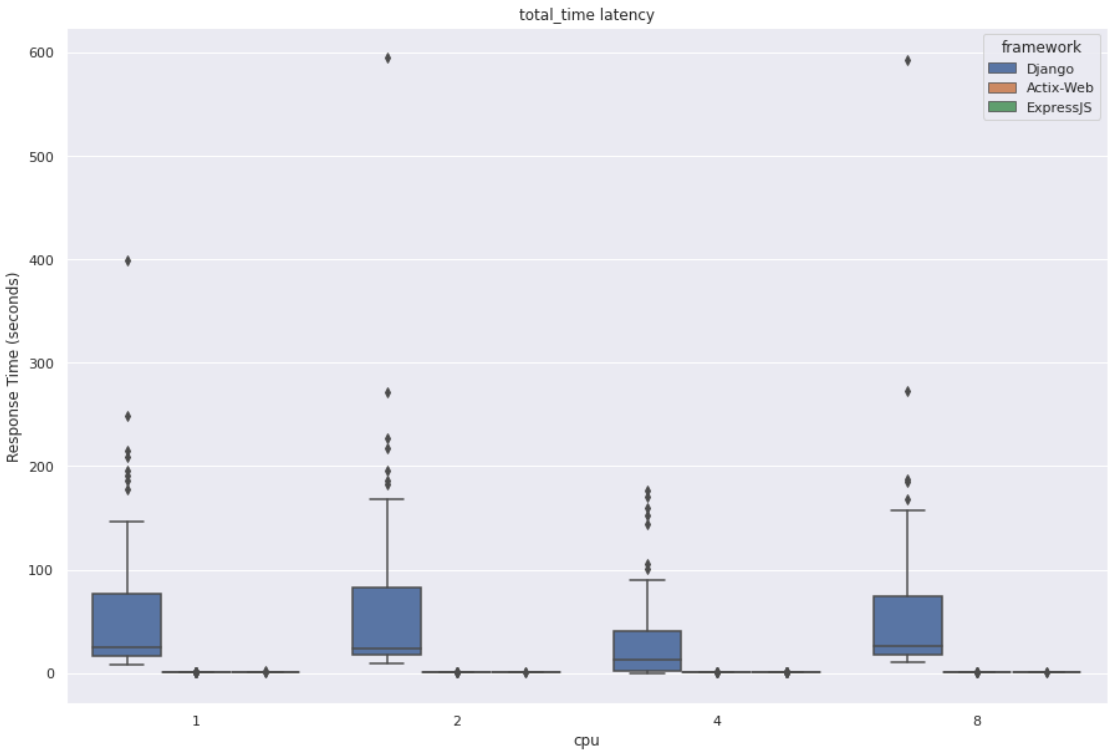

Figure 4.1: Latency Total Time Difference in CPU Threads All Framework

The Figure 4.1 show that for Django framework with increases on CPU threads there is not much significant difference in the improve of performance. This is due to `gunicorn` worker are not able to fully utilize multiple CPU cores.

This Figure 4.2 show that Express.js and Actix-Web does have improvement when their is an increase in CPU threads. It especially can be seen in the improvement in Express.js has the variance of the data is reduced, and their is a trend in improvement

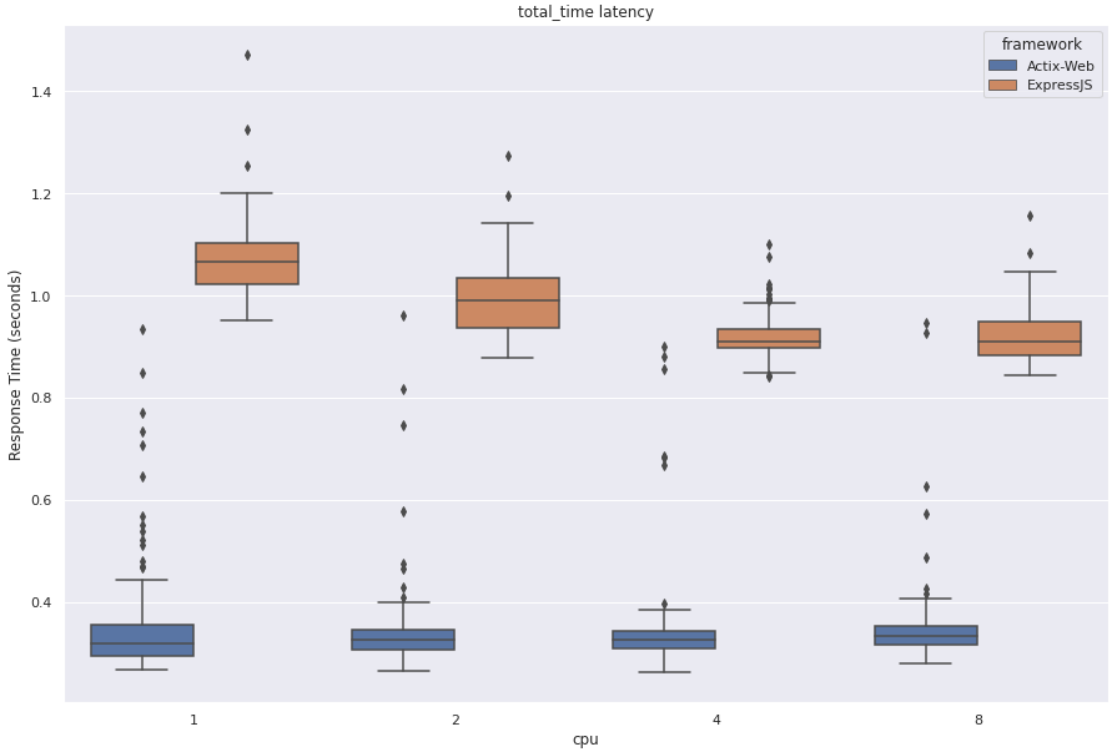

Figure 4.2: Latency Total Time Difference in CPU Threads Express.js and Actix-Web

of latency. Where as Actix-Web the outliers is greatly reduced when the CPU threads increases.

4.1.2 Throughput

The Figure 4.3 show that for Django data has an anomaly on the CPU 4 threads, this is due to the default benchmark for increases in warehouse data is done when the CPU thread is at 4. During the Delivery transaction their is a huge chance for the district on to have any delivery this causes the response time to be very low which causes many outliers that is high throughput.

This Figure 4.4 shows that as the number of threads increases the variance of the Actix-Web reduces and the outliers also reduce. As for Express.js you can see the trend of throughput increases with the number of the thread increases.

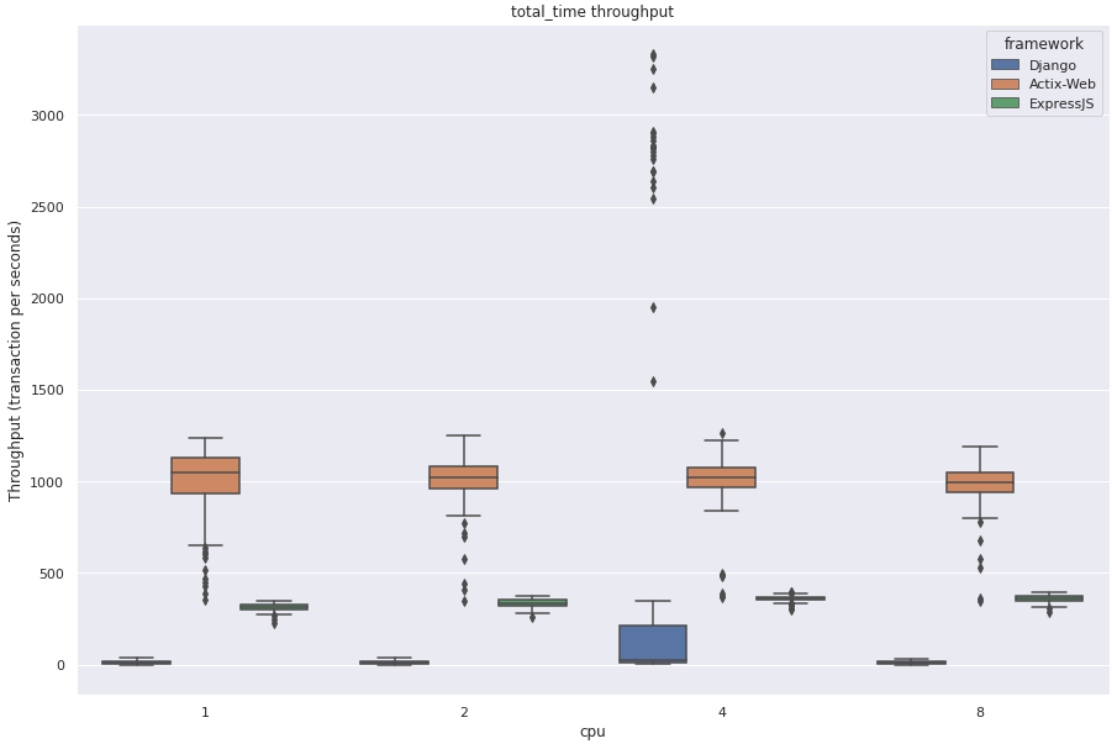

Figure 4.3: Throughput Total Time Difference in CPU Threads All Framework

4.2 Difference in Number of Main Memory

4.2.1 Latency

This Figure 4.5 shows that as the RAM size increase Django latency reduces in both total latency, and the variance of the data is also reduced.

This Figure 4.6 show that Express.js and Actix-Web does have improvement when their is an increase in RAM sizes. The overall improvement is no as significant as the reduction in data variance.

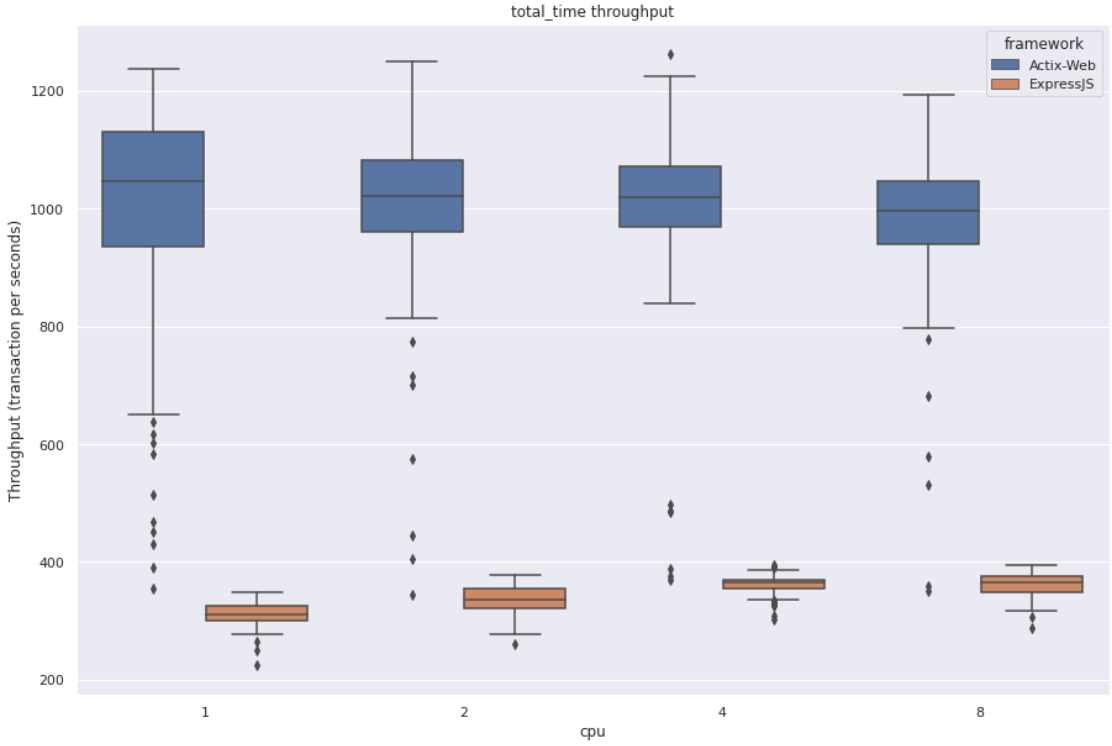

Figure 4.4: Throughput Total Time Difference in CPU Threads Express.js and Actix-Web

4.2.2 Throughput

Both Figure 4.7 and Figure 4.8 shows that as the amount of RAM size increase the performance gets better, as you can see by the trend. The reason Django has high throughput at 8 GB of RAM is due the probability of the Delivery transactions to be skipped due to there is no delivery on the given district. The increase in performance contribute to the amount of main memory the framework will have to utilize which causes less IO overhead.

4.3 Difference in Data Amount

4.3.1 Latency

This Figure 4.9 shows that as the Warehouse size increase Django latency slightly increase. The reason the trend is not more dominant it due to lack of sample size.

This Figure 4.10 show that Express.js and Actix-Web does cause the latency to slow down as the number of Warehouses increases. This is simply due to the CRUD

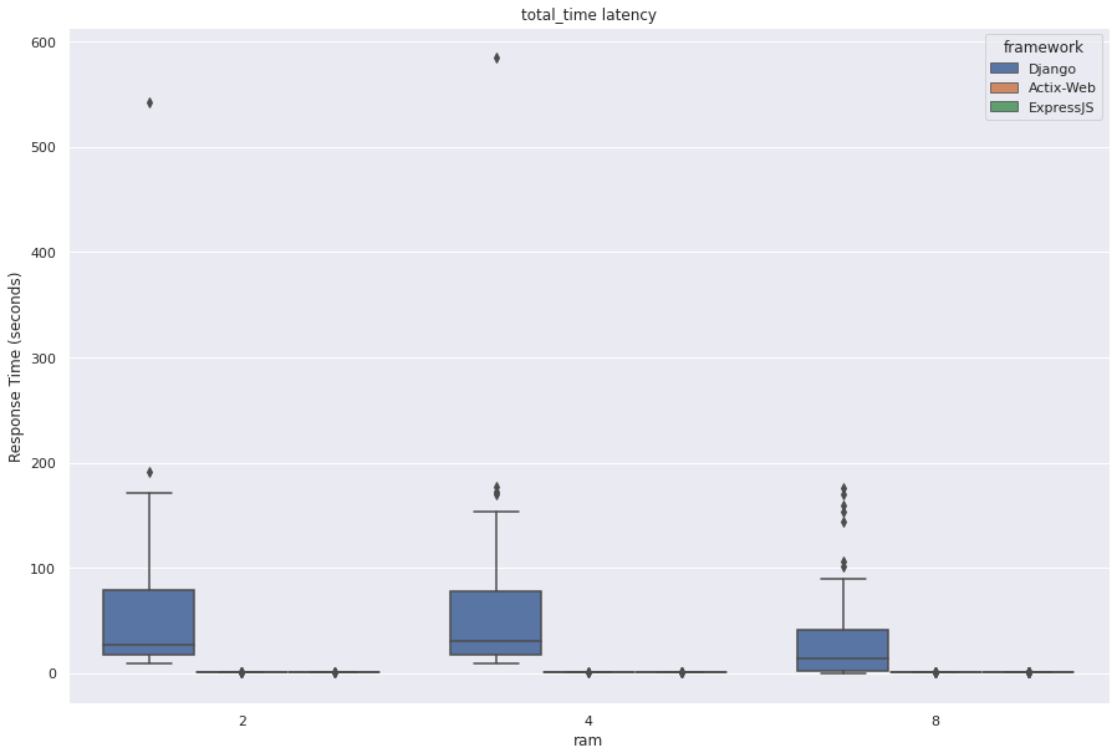

Figure 4.5: Latency Total Time Difference in RAM All Framework

database needs to do, and the querying management of the framework. As the number of Warehouse increase it is reasonable that the response time will also be slower.

4.3.2 Throughput

Both Figure 4.11 and Figure 4.12 shows that there is not a significant difference in throughput as the number of Warehouse increases. There is a slight decrease in throughput of Actix-Web, but the data variance also greatly increases. We can conclude that to determine a trend more data sample will be needed.

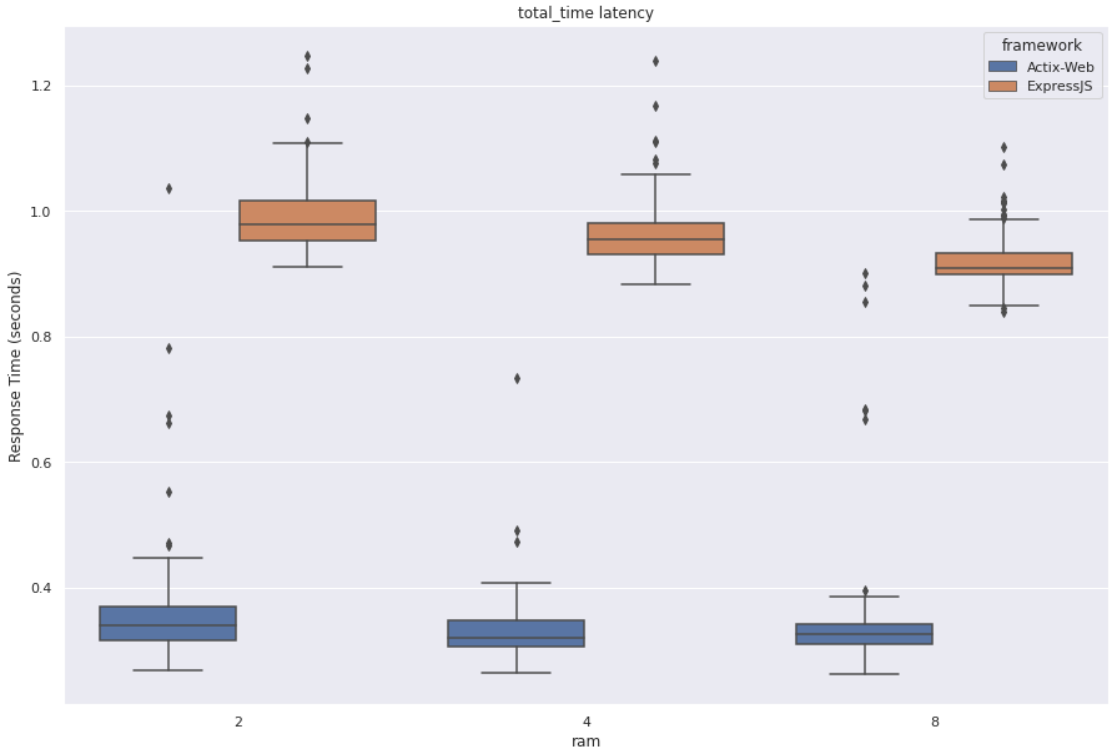

Figure 4.6: Latency Total Time Difference in RAM Express.js and Actix-Web

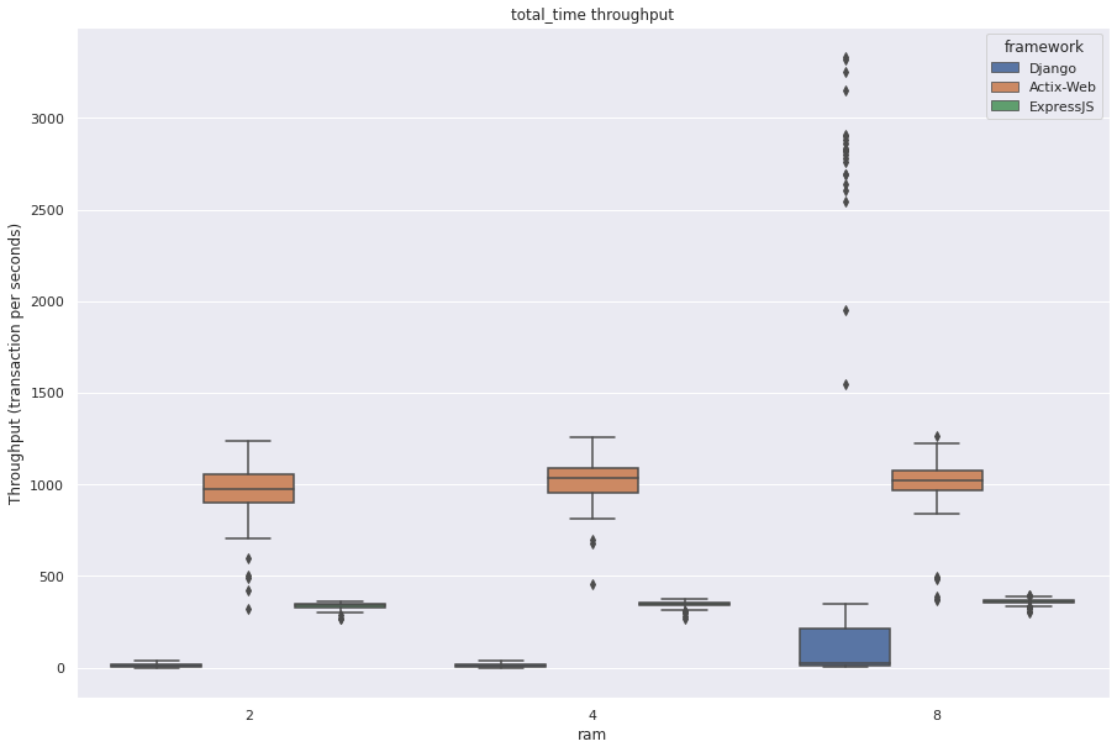

Figure 4.7: Throughput Total Time Difference in RAM All Framework

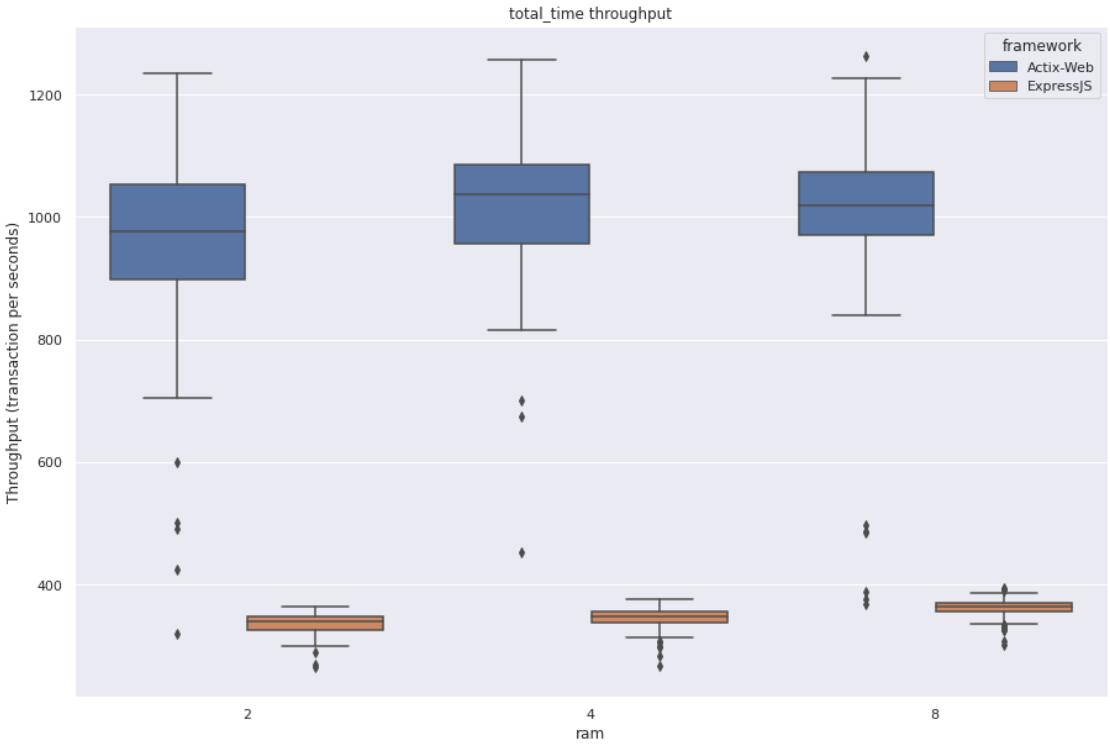

Figure 4.8: Throughput Total Time Difference in RAM Express.js and Actix-Web

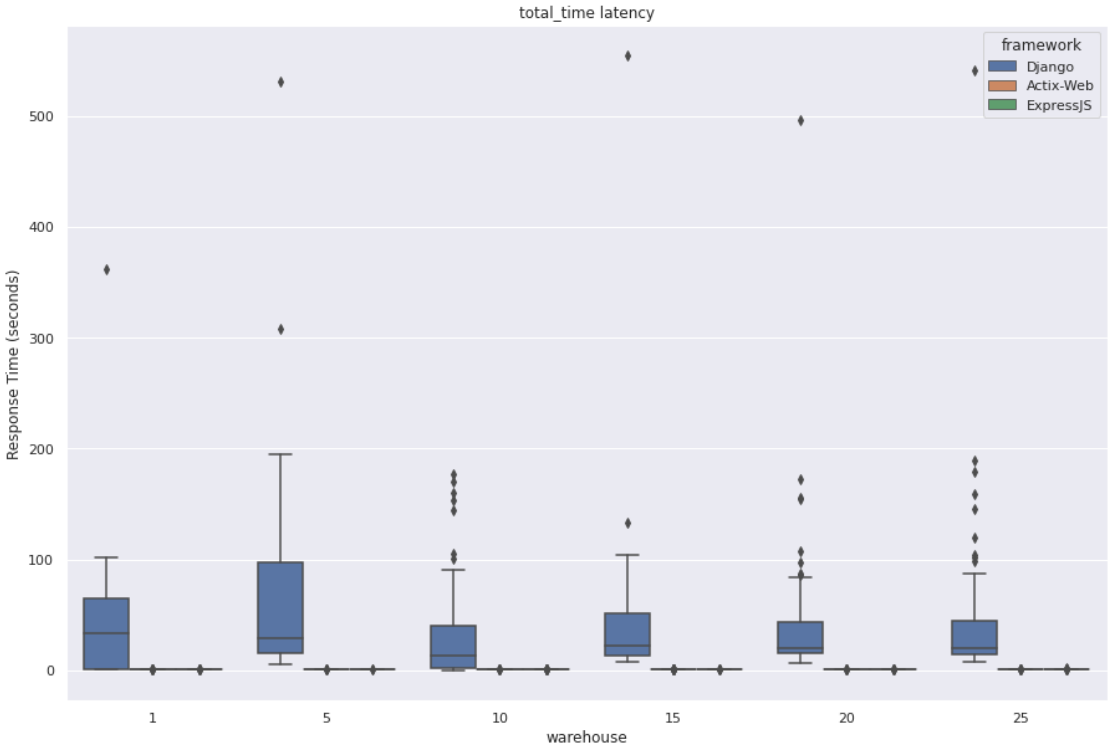

Figure 4.9: Latency Total Time Difference in Warehouse All Framework

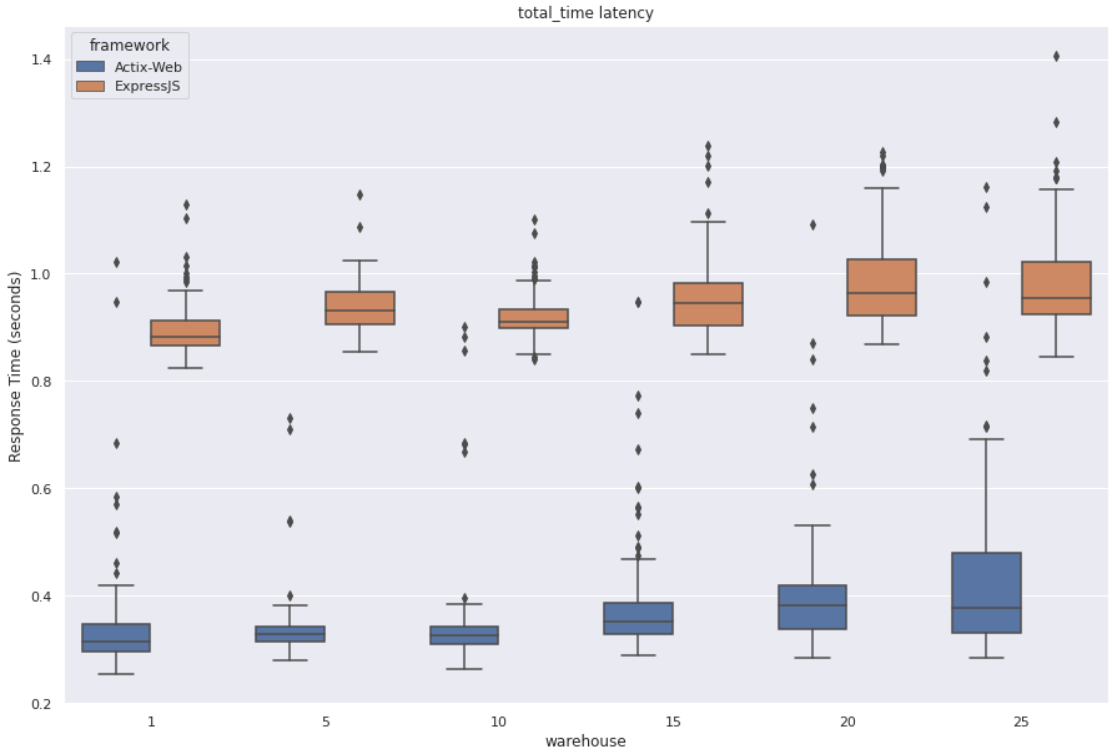

Figure 4.10: Latency Total Time Difference in Warehouse Express.js and Actix-Web

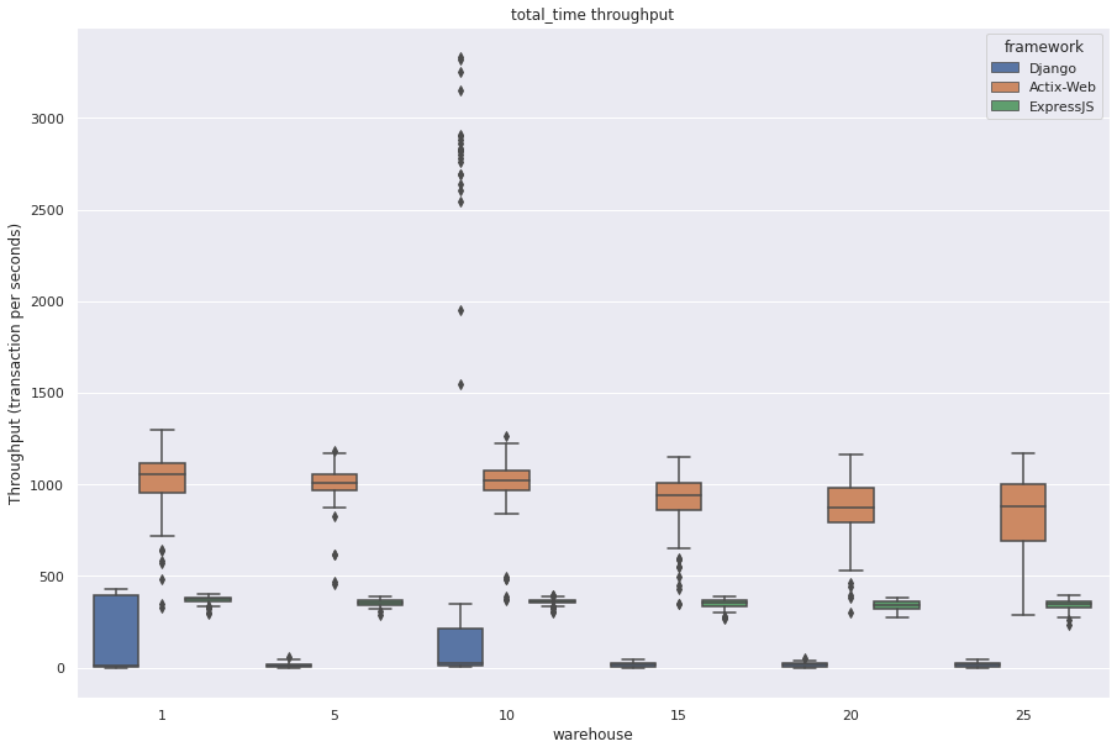

Figure 4.11: Throughput Total Time Difference in Warehouse All Framework

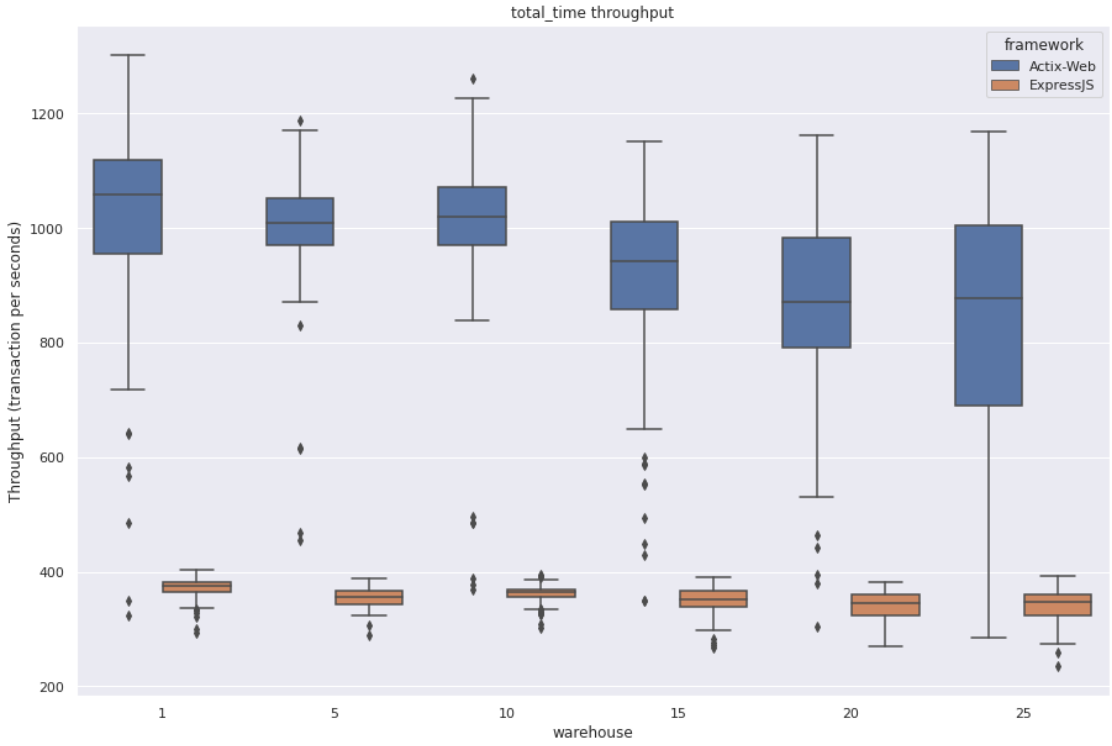

Figure 4.12: Throughput Total Time Difference in Warehouse Express.js and Actix-Web

CHAPTER 5

QUALITATIVE ANALYSIS

As a project manager it is crucial to pick a framework which will fulfil the project requirement, but it is also important for the framework to be easy to code and setup. These are my quantitative analysis of each framework on how easy is it to code, how are the documentation for each framework, and how are is the language or framework learning curve.

5.1 Ease of Code

It is crucial for the project manager to pick a framework in which the language is intuitively understandable, and easily can be learn or customize. Hence it is crucial that the programming language behind the framework will also pay an important role.

5.1.1 Django

Django web framework is based on Python programming language. Which an interpreted language, dynamically typed, and garbage-collected [10]. It core philosophy are

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts

The syntax also promotes readability. This great helps in development of Django. As it is very easy to understand, get new developer up to speed, and lower the barrier of entries.

5.1.2 Express.js

Express.js web framework is based on Node.js which is based on JavaScript programming language. JavaScript was created to run on Netscape browser, but the author also wants a rich, complied, and professional language feature like Java. As HTML 5 emerges it greatly helps in the boosting the popularity of the language [8]. With project such Node.js it allows frontend developers to use JavaScript was a server language also.

With a rich history behind the language, and the wide spread use of the language for both frontend development, and backend development with the help of Node.js. It ensure that they are plenty of developer available, or the skill is easily transferable from complied languages such as Java, or C++. This both helps old developer, new developer to understand the language much better.

5.1.3 Actix-Web

Actix-Web web framework is based on Rust programming language. Which is a compiled language with ensured memory safety. Rust was created to fill the gap of low level control of C but with high functioning capability of C++ with memory safety [4]. Rust is also a new language which comes with its package manager `cargo` which allows dependencies management and tools for linting and formatting code. This helps greatly during development cycle to put everyone on the same track on syntax of the code, and remove problem of dependencies conflict. Rust also is great at error management, and dealing with Null pointer. As Rust is a memory safe when compiled they will not be crashes due to memory leaks. Rust also provide great ability for functional programmer with it's matching statement, currying, iterator, maps, folds etc. This great help developer to to focus on coding in a strict memory safe way with flexibility of coding style.

5.2 Documentation

It is important for the project manager to pick a framework in which the framework documentation is comprehensive, and easily find the information needed. It is also

important for the framework to have other support such as questions in forums, or blog which explains or summaries a feature of a framework.

5.2.1 Django

As Python is seen to be a beginner language there is great amount of documentation and other supporting material to help developer in solving their problem. This perk also carried it way to Django web framework which has a very comprehensive guide on development of the web framework, very good support information such as blogs or YouTube guides. This helps developer both new and experience in learning, or revisiting sections of the documentation.

5.2.2 Express.js

As JavaScript is one of the oldest stable language in theses chosen frameworks, it has a rich history behind it. With it initial focus was on frontend development it already had good following and community behind the language. Later with the introduction of Node.js the development of server was able to achieved easily with JavaScript. This great help JavaScript as the amount of community guide, and history of documentation and familiarity of it is already built. This allow experienced developer to intuitively search for the needed information easily. It also help new developer greatly with it's abundance of guides, tutorial, and forum. With wide style of information it allows new developer to pick an choose the their best way to learn the language and the framework.

5.2.3 Actix-Web

As Rust is one the newest language in theses chosen frameworks, it had great features and fixes that major programming languages faces. Due to the memory safety approach of Rust it a core fundamental of the language, the documentation. Rust provides in depth information about each part of their fundamental from Rust Book which discuss general understanding of Rust language, Rust By Example which provides example on how to do certain things in the language with in depth explanation on what should be

used during production, and lastly Rustlings which is basic exercises which will drill you on understand Rust syntax and its convenient features such as maps, iterator, folding, error handling, matching statements, structs etc. This modernization of the language help developer from both old and new to quickly grasp the language in a short time. It does not matter if you are familiar with low level language such as C or more memory or object handling language such as C++. With Rust documentation you will feel just like home with Rust. Theses documentation also lends its hand to its package manager `cargo` which has one of the best documentation ability to query for the information you need. The framework also provide many example on how you accomplish simple task then how to adapt it for more complex task with examples. We would say that Actix-Web has the best documentation of all the web frameworks.

5.3 Learning Curve

It is crucial for project manager to pick a framework with great mix of developer and great mix of skill. Hence it is important that the learning curve of the web framework, and the language needs to be manageable.

5.3.1 Django

As Python being on the best beginner language, it great help to reduce the learning curve. Django also adopt most of the Python philosophy in their web framework development. This helps Django to also have a lowest lowering curve comparing to the selected web framework.

5.3.2 Express.js

As JavaScript has one of the richest history comparing to the selected framework, it has greatest support and ability to transfer the information is high. This idea also exist in Express.js which allows experienced developer from other framework to easily transfer their knowledge to use Express.js with the least friction.

5.3.3 Actix-Web

As Rust is one of the newest language comparing to the the selected framework. It has many great modern language features, but it strictness of memory safety and unusual syntax. Actix-Web was develop with Actor design pattern in mind which for developer who is familiar with the design pattern it will be a easy transfer, but for developer who are not familiar with the design pattern it will increase learning curve. Hence Actix-Web has the highest learning curve in the select web frameworks, due to it strictness of language and Actor design pattern.

CHAPTER 6

QUANTITATIVE ANALYSIS

6.1 Development Timeline

The development is life cycle is greatly important for the project, hence it is important for the project manager to pick a web framework which suites the project goals the most. Having a consistent development time is more important than having a wide range of development time, as it will help the timeline projection and features importance.

This section explains the development time of each application programming interface for each framework. The goal of each web framework is to emulation the TPC-C benchmark. We first develop the emulation firstly in Django, then Actix-Web, then Express.js. Hence most of the problem with replicating the emulation was fixed in Django framework, which may cause development time to longer than it needed.

6.1.1 Django

The development time for Django is 5 days, where the first 2 day was spent interacting with database, and experimenting simple and complex CRUD. On the 3th day was spent implementing New Order transaction, and Payment transaction. On the last two day was spent implementing the rest of the transactions, and check scalability. Majority of the development time is spent in reading the TPC-C documentation and implementing it. Logging of Django also greatly help to find error, or bug in the code.

6.1.2 Express.js

The development time for Express.js is 1 day, where the first few hours was spent interacting with database, and experimenting simple and complex CRUD. Later hours was spent creating transactions, and configuring for production. Majority of the time was spent on dealing with complex query, and logging to see the correctness of emulation.

6.1.3 Actix-Web

The development time for Express.js is a week, where the 3 days was spent interacting with database, and experimenting simple and complex CRUD. On the 4th day was spent understanding Actor design pattern, and configure all the libraries. On the last days the time was spent implementing all the transaction, and do error handling.

CHAPTER 7

ARTIFACTS

These are scripts used in the benchmark for setting up the virtual machines for the database, and the web frameworks. This also includes data collection scripts. Here is the repository containing all the code used: `https://github.com/narula2000/thesis-project`

7.1 Generate Database Virtual Machine

```
#!/bin/bash

virt-install \
  --name database \
  --vcpus 4 \
  --ram 8192 \
  --disk path=db.qcow2,size=85 \
  --console pty,target_type=serial \
  --cdrom ubuntu-20.04.4-live-server-amd64.iso2

```

7.2 Destroy Database Virtual Machine

```
#!/bin/bash

virsh list --all
virsh shutdown database
virsh destroy database
virsh undefine database --remove-all-storage
virsh list --all

```

7.3 Setup Database Virtual Machine

We change the ip and warehouse according to the ip of the virtual machines, and warehouse to the benchmark criteria. Create data is a script which generate data according to TPC-C requirement.

```
#!/bin/bash

warehouse=10

BGREEN='\033[1;32m'

echo -e "${BGREEN}----------------------> ${BGREEN}Copy Config"
cp tmux.conf ~/.tmux.conf
cp vimrc ~/.vimrc

echo -e "${BGREEN}----------------------> ${BGREEN}Edit Bashrc"
echo '' >> ~/.bashrc
echo 'ip="192.168.122."' >> ~/.bashrc
echo 'dbip="192.168.122."' >> ~/.bashrc
echo '' >> ~/.bashrc
echo 'export ip' >> ~/.bashrc
echo 'export dbip' >> ~/.bashrc
echo '' >> ~/.bashrc
echo 'alias check-ip="virsh domifaddr ubuntu"' >> ~/.bashrc
echo 'alias check-ip-db="virsh domifaddr database"' >> ~/.bashrc
echo 'alias vm="ssh vm@${ip}"' >> ~/.bashrc
echo 'alias db="ssh vm@${dbip}"' >> ~/.bashrc
echo 'alias ee="vim ."' >> ~/.bashrc
echo 'alias e="vim"' >> ~/.bashrc
echo 'alias ss="source ~/.bashrc"' >> ~/.bashrc

echo -e "${BGREEN}----------------------> ${BGREEN}Install Postgres"
sudo apt update && sudo apt upgrade -y
sudo apt install build-essential -y postgresql-client-common
postgresql libpq-dev -y
sudo systemctl enable ssh --now

echo -e "${BGREEN}----------------------> ${BGREEN}Config Postgres"
sudo rm /etc/postgresql/12/main/pg_hba.conf
sudo cp postgres-config.conf /etc/postgresql/12/main/pg_hba.conf
sudo bash -c "echo \"listen_addresses = '*'\" >> /etc/postgresql
/12/main/postgresql.conf"
sudo bash -c "echo \"log_statement = 'ddl'\" >> /etc/postgresql
/12/main/postgresql.conf"
sudo bash -c "echo \"max_wal_size = 5GB\" >> /etc/postgresql/12/
main/postgresql.conf"

echo -e "${BGREEN}----------------------> ${BGREEN}Start Postgres"
sudo pg_ctlcluster 12 main start
sudo service postgresql restart

echo -e "${BGREEN}----------------------> ${BGREEN}Gen Data"
cd create-data
bash create-data.sh $warehouse

echo -e "${BGREEN}----------------------> ${BGREEN}Gen DB"
cd ..
bash gen-db.sh

```

This is the `gen-db.sh`:

```
#!/bin/bash

line=`cat /home/vm/database-data/history.csv | wc -l`
echo $line

set -e

psql -U postgres <<-EOSQL

    DROP TABLE IF EXISTS "warehouse" CASCADE;
    DROP TABLE IF EXISTS "district" CASCADE;
    DROP TABLE IF EXISTS "item" CASCADE;
    DROP TABLE IF EXISTS "customer" CASCADE;
    DROP TABLE IF EXISTS "history" CASCADE;
    DROP TABLE IF EXISTS "stock" CASCADE;
    DROP TABLE IF EXISTS "orders" CASCADE;
    DROP TABLE IF EXISTS "new_order" CASCADE;
    DROP TABLE IF EXISTS "order_line" CASCADE;

    CREATE TABLE WAREHOUSE (
      W_ID SMALLINT DEFAULT '0' NOT NULL,
      W_NAME VARCHAR(16) DEFAULT NULL,
      W_STREET_1 VARCHAR(32) DEFAULT NULL,
      W_STREET_2 VARCHAR(32) DEFAULT NULL,
      W_CITY VARCHAR(32) DEFAULT NULL,
      W_STATE VARCHAR(2) DEFAULT NULL,
      W_ZIP VARCHAR(9) DEFAULT NULL,
      W_TAX FLOAT DEFAULT NULL,
      W_YTD FLOAT DEFAULT NULL,
      CONSTRAINT W_PK_ARRAY PRIMARY KEY (W_ID)
    );

    CREATE TABLE DISTRICT (
      D_ID SMALLINT DEFAULT '0' NOT NULL,
      D_W_ID SMALLINT DEFAULT '0' NOT NULL REFERENCES WAREHOUSE (W_ID),
      D_NAME VARCHAR(16) DEFAULT NULL,
      D_STREET_1 VARCHAR(32) DEFAULT NULL,
      D_STREET_2 VARCHAR(32) DEFAULT NULL,
      D_CITY VARCHAR(32) DEFAULT NULL,
      D_STATE VARCHAR(2) DEFAULT NULL,
      D_ZIP VARCHAR(9) DEFAULT NULL,
      D_TAX FLOAT DEFAULT NULL,
      D_YTD FLOAT DEFAULT NULL,
      D_NEXT_O_ID INT DEFAULT NULL,
      PRIMARY KEY (D_W_ID,D_ID)
    );

    CREATE TABLE ITEM (
      I_ID INTEGER DEFAULT '0' NOT NULL,
      I_IM_ID INTEGER DEFAULT NULL,
      I_NAME VARCHAR(32) DEFAULT NULL,
      I_PRICE FLOAT DEFAULT NULL,
      I_DATA VARCHAR(64) DEFAULT NULL,
```

```
54            CONSTRAINT I_PK_ARRAY PRIMARY KEY (I_ID)
55          );
56
57          CREATE TABLE CUSTOMER (
58            C_ID INTEGER DEFAULT '0' NOT NULL,
59            C_D_ID SMALLINT DEFAULT '0' NOT NULL,
60            C_W_ID SMALLINT DEFAULT '0' NOT NULL,
61            C_FIRST VARCHAR(32) DEFAULT NULL,
62            C_MIDDLE VARCHAR(2) DEFAULT NULL,
63            C_LAST VARCHAR(32) DEFAULT NULL,
64            C_STREET_1 VARCHAR(32) DEFAULT NULL,
65            C_STREET_2 VARCHAR(32) DEFAULT NULL,
66            C_CITY VARCHAR(32) DEFAULT NULL,
67            C_STATE VARCHAR(2) DEFAULT NULL,
68            C_ZIP VARCHAR(9) DEFAULT NULL,
69            C_PHONE VARCHAR(32) DEFAULT NULL,
70            C_SINCE TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
71            C_CREDIT VARCHAR(2) DEFAULT NULL,
72            C_CREDIT_LIM FLOAT DEFAULT NULL,
73            C_DISCOUNT FLOAT DEFAULT NULL,
74            C_BALANCE FLOAT DEFAULT NULL,
75            C_YTD_PAYMENT FLOAT DEFAULT NULL,
76            C_PAYMENT_CNT INTEGER DEFAULT NULL,
77            C_DELIVERY_CNT INTEGER DEFAULT NULL,
78            C_DATA VARCHAR(500),
79            PRIMARY KEY (C_W_ID,C_D_ID,C_ID),
80            UNIQUE (C_W_ID,C_D_ID,C_LAST,C_FIRST),
81            CONSTRAINT C_FKEY_D FOREIGN KEY (C_D_ID, C_W_ID) REFERENCES
     DISTRICT (D_ID, D_W_ID)
82          );
83          CREATE INDEX IDX_CUSTOMER ON CUSTOMER (C_W_ID,C_D_ID,C_LAST);
84
85          CREATE TABLE HISTORY (
86            H_ID SERIAL PRIMARY KEY,
87            H_C_ID INTEGER DEFAULT NULL,
88            H_C_D_ID SMALLINT DEFAULT NULL,
89            H_C_W_ID SMALLINT DEFAULT NULL,
90            H_D_ID SMALLINT DEFAULT NULL,
91            H_W_ID SMALLINT DEFAULT '0' NOT NULL,
92            H_DATE TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
93            H_AMOUNT FLOAT DEFAULT NULL,
94            H_DATA VARCHAR(32) DEFAULT NULL,
95            CONSTRAINT H_FKEY_C FOREIGN KEY (H_C_ID, H_C_D_ID, H_C_W_ID
    ) REFERENCES CUSTOMER (C_ID, C_D_ID, C_W_ID),
96            CONSTRAINT H_FKEY_D FOREIGN KEY (H_D_ID, H_W_ID) REFERENCES
     DISTRICT (D_ID, D_W_ID)
97          );
98
99          CREATE TABLE STOCK (
100           S_I_ID INTEGER DEFAULT '0' NOT NULL REFERENCES ITEM (I_ID),
101           S_W_ID SMALLINT DEFAULT '0 ' NOT NULL REFERENCES WAREHOUSE
    (W_ID),
102           S_QUANTITY INTEGER DEFAULT '0' NOT NULL,
103           S_DIST_01 VARCHAR(32) DEFAULT NULL,
104           S_DIST_02 VARCHAR(32) DEFAULT NULL,
```

```
105             S_DIST_03 VARCHAR(32) DEFAULT NULL,
106             S_DIST_04 VARCHAR(32) DEFAULT NULL,
107             S_DIST_05 VARCHAR(32) DEFAULT NULL,
108             S_DIST_06 VARCHAR(32) DEFAULT NULL,
109             S_DIST_07 VARCHAR(32) DEFAULT NULL,
110             S_DIST_08 VARCHAR(32) DEFAULT NULL,
111             S_DIST_09 VARCHAR(32) DEFAULT NULL,
112             S_DIST_10 VARCHAR(32) DEFAULT NULL,
113             S_YTD INTEGER DEFAULT NULL,
114             S_ORDER_CNT INTEGER DEFAULT NULL,
115             S_REMOTE_CNT INTEGER DEFAULT NULL,
116             S_DATA VARCHAR(64) DEFAULT NULL,
117             PRIMARY KEY (S_W_ID, S_I_ID)
118         );
119
120         CREATE TABLE ORDERS (
121             O_ID INTEGER DEFAULT '0' NOT NULL,
122             O_D_ID SMALLINT DEFAULT '0' NOT NULL,
123             O_W_ID SMALLINT DEFAULT '0' NOT NULL,
124             O_C_ID INTEGER DEFAULT NULL,
125             O_ENTRY_D TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
126             O_CARRIER_ID INTEGER DEFAULT NULL,
127             O_OL_CNT INTEGER DEFAULT NULL,
128             O_ALL_LOCAL INTEGER DEFAULT NULL,
129             PRIMARY KEY (O_W_ID, O_D_ID, O_ID),
130             UNIQUE (O_W_ID, O_D_ID, O_C_ID, O_ID),
131             CONSTRAINT O_FKEY_C FOREIGN KEY (O_C_ID, O_D_ID, O_W_ID)
    REFERENCES CUSTOMER (C_ID, C_D_ID, C_W_ID)
132         );
133         CREATE INDEX IDX_ORDERS ON ORDERS (O_W_ID, O_D_ID, O_C_ID);
134
135         CREATE TABLE NEW_ORDER (
136             NO_O_ID INTEGER DEFAULT '0' NOT NULL,
137             NO_D_ID SMALLINT DEFAULT '0' NOT NULL,
138             NO_W_ID SMALLINT DEFAULT '0' NOT NULL,
139             CONSTRAINT NO_PK_TREE PRIMARY KEY (NO_D_ID, NO_W_ID, NO_O_ID)
    ,
140             CONSTRAINT NO_FKEY_O FOREIGN KEY (NO_O_ID, NO_D_ID, NO_W_ID
    ) REFERENCES ORDERS (O_ID, O_D_ID, O_W_ID)
141         );
142
143         CREATE TABLE ORDER_LINE (
144             OL_O_ID INTEGER DEFAULT '0' NOT NULL,
145             OL_D_ID SMALLINT DEFAULT '0' NOT NULL,
146             OL_W_ID SMALLINT DEFAULT '0' NOT NULL,
147             OL_NUMBER INTEGER DEFAULT '0' NOT NULL,
148             OL_I_ID INTEGER DEFAULT NULL,
149             OL_SUPPLY_W_ID SMALLINT DEFAULT NULL,
150             OL_DELIVERY_D TIMESTAMP NULL DEFAULT NULL,
151             OL_QUANTITY INTEGER DEFAULT NULL,
152             OL_AMOUNT FLOAT DEFAULT NULL,
153             OL_DIST_INFO VARCHAR(32) DEFAULT NULL,
154             PRIMARY KEY (OL_W_ID, OL_D_ID, OL_O_ID, OL_NUMBER),
155             CONSTRAINT OL_FKEY_O FOREIGN KEY (OL_O_ID, OL_D_ID, OL_W_ID
    ) REFERENCES ORDERS (O_ID, O_D_ID, O_W_ID),
```

```
156         CONSTRAINT OL_FKEY_S FOREIGN KEY (OL_I_ID, OL_SUPPLY_W_ID)
    REFERENCES STOCK (S_I_ID, S_W_ID)
157     );
158     CREATE INDEX IDX_ORDER_LINE_TREE ON ORDER_LINE (OL_W_ID,
    OL_D_ID,OL_O_ID);
159
160     COPY WAREHOUSE FROM '/home/vm/database-data/warehouse.csv'
    WITH (FORMAT CSV, DELIMITER ',', NULL "NULL");
161     COPY DISTRICT FROM '/home/vm/database-data/district.csv' WITH
     (FORMAT CSV, DELIMITER ',', NULL "NULL");
162     COPY ITEM FROM '/home/vm/database-data/item.csv' WITH (FORMAT
     CSV, DELIMITER ',', NULL "NULL");
163     COPY CUSTOMER FROM '/home/vm/database-data/customer.csv' WITH
     (FORMAT CSV, DELIMITER ',', NULL "NULL");
164     COPY HISTORY FROM '/home/vm/database-data/history.csv' WITH (
    FORMAT CSV, DELIMITER ',', NULL "NULL");
165     COPY STOCK FROM '/home/vm/database-data/stock.csv' WITH (
    FORMAT CSV, DELIMITER ',', NULL "NULL");
166     COPY ORDERS FROM '/home/vm/database-data/order.csv' WITH (
    FORMAT CSV, DELIMITER ',', NULL "NULL");
167     COPY NEW_ORDER FROM '/home/vm/database-data/new_order.csv'
    WITH (FORMAT CSV, DELIMITER ',', NULL "NULL");
168     COPY ORDER_LINE FROM '/home/vm/database-data/order_line.csv'
    WITH (FORMAT CSV, DELIMITER ',', NULL "NULL");
169
170     CREATE SEQUENCE H_ID_SEQ OWNED BY HISTORY.H_ID;
171     SELECT SETVAL('H_ID_SEQ', ${line});
172     ALTER TABLE HISTORY ALTER COLUMN H_ID SET DEFAULT nextval('
    H_ID_SEQ');
173 EOSQL
174
```

7.4 Reset Database Virtual Machine

```
1  #!/bin/bash
2
3  warehouse=10
4
5  BGREEN='\033[1;32m'
6  echo -e "${BGREEN}----------> ${BGREEN}Gen Data"
7  cd create-data
8  bash create-data.sh $warehouse
9
10 echo -e "${BGREEN}----------> ${BGREEN}Gen DB"
11 cd ..
12 bash gen-db.sh
13
```

7.5 Generate Framework Virtual Machine

We will vary the `vcups`, and `ram` according to benchmark criteria.

```
#!/bin/bash

virt-install \
  --name ubuntu \
  --vcpus 4 \
  --ram 8192 \
  --disk path=u20.qcow2,size=15 \
  --console pty,target_type=serial \
  --cdrom ubuntu-20.04.4-live-server-amd64.iso

```

7.6 Destroy Framework Virtual Machine

```
#!/bin/bash

virsh list --all
virsh shutdown ubuntu
virsh destroy ubuntu
virsh undefine ubuntu --remove-all-storage
virsh list --all

```

7.7 Setup Framework Virtual Machine

```
#!/bin/bash

BGREEN='\033[1;32m'

echo -e "${BGREEN}----------------------> ${BGREEN}Copy Config"
cp tmux.conf ~/.tmux.conf
cp vimrc ~/.vimrc

echo -e "${BGREEN}----------------------> ${BGREEN}Edit Bashrc"
echo '' >> ~/.bashrc
echo 'ip="192.168.122."' >> ~/.bashrc
echo 'dbip="192.168.122."' >> ~/.bashrc
echo '' >> ~/.bashrc
echo 'export ip' >> ~/.bashrc
echo 'export dbip' >> ~/.bashrc
echo '' >> ~/.bashrc
echo 'alias check-ip="virsh domifaddr ubuntu"' >> ~/.bashrc
echo 'alias check-ip-db="virsh domifaddr database"' >> ~/.bashrc
echo 'alias vm="ssh vm@${ip}"' >> ~/.bashrc
echo 'alias db="ssh vm@${dbip}"' >> ~/.bashrc
echo 'alias ee="vim ."' >> ~/.bashrc
echo 'alias e="vim"' >> ~/.bashrc
echo 'alias ss="source ~/.bashrc"' >> ~/.bashrc

echo -e "${BGREEN}----------------------> ${BGREEN}Install Reqs"
sudo apt update && sudo apt upgrade -y
sudo apt install build-essential libpq-dev -y
```

```
    sudo systemctl enable ssh --now

    echo -e "${BGREEN}----------------------> ${BGREEN}Install API Reqs"
    cd server
    bash get-req.sh

```

Where `get-req.sh` installs the dependency needed to run the application programming interface. Here is the scripts to install dependency for each framework.

7.7.1 Django

```
    #!/bin/bash

    sudo apt update && sudo apt upgrade -y
    sudo apt install software-properties-common -y
    sudo apt install python3-pip -y
    pip install gunicorn django psycopg2-binary djangorestframework
    django-stubs django-extensions

```

7.7.2 Express.js

```
    #!/bin/bash

    sudo apt update && sudo apt upgrade -y
    sudo apt install build-essential -y
    curl -sL https://deb.nodesource.com/setup_16.x -o /tmp/
    nodesource_setup.sh
    sudo bash /tmp/nodesource_setup.sh
    sudo apt install nodejs -y

```

7.7.3 Actix-Web

```
    #!/bin/bash

    sudo apt update && sudo apt upgrade -y
    sudo apt install build-essential -y
    curl https://sh.rustup.rs -sSf | sh -s -- --profile default

```

7.8 Data Collection

```
    import csv
    import requests
    import os

```

```
    ip = os.getenv('ip', "127.0.0.1")

    def get_req_time(api, transaction):
        url = f'{api}/{transaction}'
        res = requests.get(url, timeout=5000)
        return res.elapsed.total_seconds()

    def make_csv(headers, data, fname='out.csv',):
        with open(fname, 'w') as file:
            write = csv.writer(file)
            write.writerow(headers)
            for tup in zip(*data):
                print("-->", tup)
                write.writerow(tup)

    if __name__ == '__main__':
        query_amount = 100
        fname = 'out.csv'
        api = f"http://{ip}:8000/api"
        headers = ["new_order", "payment", "order_status", "delivery"
    , "stock_level", "total_time"]
        transactions = headers[:-1]
        times = []
        for idx in range(query_amount):
            req_times = []
            for transaction in transactions:
                req_times.append(get_req_time(api, transaction))
            req_times.append(sum(req_times))
            print("-> total_time", "\t",idx, "\t",req_times[-1])
            times.append(tuple(req_times))

        new_orders = (t[0] for t in times)
        payments = (t[1] for t in times)
        order_statuses = (t[2] for t in times)
        deliverys = (t[3] for t in times)
        stock_levels = (t[4] for t in times)
        total_time = (t[5] for t in times)

        data_rows = [new_orders, payments, order_statuses, deliverys,
     stock_levels, total_time]

        make_csv(headers, data_rows, fname)

```

CHAPTER 8

CONCLUSION

With the results given we can clearly see that performance wise the ranking goes Actix-Web, Express.js, Django accordingly. The goal of a project manager to not only pick the best performance web framework, but to pick the most suitable web framework for the current problem, this include complexity of the application, experience of the development team, and development time. Hence we can conclude our ranking as this:

- For small scale application choosing Django will greatly help in development time, and the performance will not be greatly impacted.
- For a small to medium scale application Express.js is greatly recommended as it performance and history of the language will greatly help both new and experienced developer. It will also allow the ability for the framework to be scaled
- For a medium to large scale application Actix-Web is greatly recommend as it strictness of the language and define design pattern will greatly help in the readability and maintainability of the code, and the performance is unbeatable in the chosen framework.

8.1 Lesson learn Future Plans

The lesson learn was to do wide shallow research before focusing on specific topic. This will help reduce the research and planning time greatly. This will also reduce mistake or problem that might have been faced due to lack of planning or research. The other important lesson we have learn is that stress test your work for any possible flaws before deployment, this will reduce the chance of the application or the work crashing during the deployment.

The future of the project is to build a better benchmark system for testing application programming frameworks. This can be done by introducing multi-threading request handling, web framework ability to handle denial of service attacks. If possible we would like to publish this project.

REFERENCES

[1] Joshua D Drake and John C Worsley. *Practical PostgreSQL*. " O'Reilly Media, Inc.", 2002.

[2] Yasunori Goto. Kernel-based virtual machine technology. *Fujitsu Scientific and Technical Journal*, 47(3):362–368, 2011.

[3] Adrian Holovaty and Jacob Kaplan-Moss. *The definitive guide to Django: Web development done right*. Apress, 2009.

[4] Ralf Jung. Understanding and evolving the rust programming language. 2020.

[5] Scott T Leutenegger and Daniel Dias. A modeling study of the tpc-c benchmark. *ACM Sigmod Record*, 22(2):22–31, 1993.

[6] Shing Lyu. Rest apis. In *Practical Rust Web Projects*, pages 55–102. Springer, 2021.

[7] Azat Mardan. *Pro express. js*. Springer, 2014.

[8] Charles Severance. Javascript: Designing a language in 10 days. *Computer*, 45(2):7–8, 2012.

[9] Diomidis Spinellis. Git. *IEEE software*, 29(3):100–101, 2012.

[10] Guido Van Rossum et al. Python programming language. In *USENIX annual technical conference*, volume 41, pages 1–36, 2007.

BIOGRAPHY

NAME	Mr.Vikrom Narula
DATE OF BIRTH	7th February 2000
PLACE OF BIRTH	Punjab, India
INSTITUTIONS ATTENDED	Rayong English Programme School, 2005-2017
	Certificate of Upper Secondary School (Year 12)
	Mahidol University, 2017-2022
	Bachelor of Science (Computer Science)
HOME ADDRESS	574/52 m.5 Wonderland 2, Pattaya Sai 3 Alley
	Bang Lamung, Chonburi 20150, Thailand
E-MAIL	vikrom.nar@gmail.com