Jurnal Sisfotek Global
ISSN (Online): 2721 - 3161, ISSN (Print): 2088 – 1762
DOI: http://dx.doi.org/10.38101/sisfotek.v13i1.2656
Vol. 13, No. 1, March 2023, pp. 28-34

ANALYSIS OF APPLICATION PERFORMANCE TESTING USING LOAD TESTING AND STRESS TESTING METHODS IN API SERVICE

Mokhamd Hendayun[1], Arief Ginanjar[2] & Yoan Ihsan[3]

[1,2,3] Langlangbuana University, Bandung, Indonesia, 40261
E-mail: [1]hendayun@unla.ac.id, [2]arief.ginanjar@unla.ac.id, [3]ihsanyoan11@gmail.com

ARTICLE HISTORY

Received : February 27th, 2023
Revised : March 27th, 2023
Accepted :March 30th, 2023

KEYWORDS

Performance Analysis
Performance Testing
Load Testing
Blackbox Testing
Stress Testing

ABSTRACT

Testing an application is an important thing to do in software development. However, there are some tests that are sometimes missed after application development has been completed, such as performance testing. Performance testing is a type of test to ensure software will work properly under the expected workload. When performance testing is important to ensure that the functional needs of the system are running well. To ensure that the business objectives of the system are satisfactory and according to user needs can be done using Black Box Method. After ensuring the final results meet the specifications of the system needs, performance testing can be carried out. In performance testing, a load and stress testing methods are used to test the system, so as to be able to validate system performance are correct and determine the operating capacity of the system, with load limit at the break threshold and above the break threshold. This test is carried out to help determine how software can act well when accessed by several users together with a lot of data. This study conduct analyze the behavior of the system in the server environment that currently running and then optimize the configuration of the service and server to achieve concurrent users with 500 users at a time with jmeter as a performance test tools.

1. Introduction

1.1 Background

API Service for vehicle financing submission is a service for vehicle financing request applications developed in Bandung. The API function in this application is useful as a liaison between application to database and connecting between platforms. API's play an important role in the availability and speed of data access required in applications. The application is commonly used for company operations, which means that the application will be used frequently in daily operation. With many transaction processes carried out, it's must be supported by an adequate system. Some systems often not pay attention to the performance of the system being developed, so what should be a system that helps in simplifying and speeding up a job actually does not have that impact at all due to slow application performance.

Performance testing on an application system is essential to find an error. This test can be said to be good because it has the possibility to find a lack that not revealed in functional test. A successful test when the test can unpack an error that was not originally found. The main purpose of this testing it can systematically dismantle the types of errors with minimum effort and time. Based on the problems above, it is necessary to test and measure how many limits on users who can access the process in the same time simultaneously parallel, as well as resource consumption when carrying out this processes with a certain number of concurrent users, so that developers can perform tuning up configuration related to the performance of an application. Techniques that can be used to conduct testing are load and stress testing using the Black Box method. Research that specifically studies application performance is still very rare, therefore researchers hope that this paper will become one of the ongoing studies for further research.

1.2 Research Objectives

Purpose of this research is to analyze the API service performance to find out the time and resource consumption, conditions and ability of the service to handle requests simultaneously with certain concurrent users. Analysis is carried out by evaluating loading times, server resource usage, and errors found

based on service responses and service logs so that they can provide an appropriate configuration model to optimize system performance [1]. This study produce configuration model based on the test results; create a pros and cons analysis of the errors found. In addition, service performance analysis is also carried out and create the server environment had a recovery capabilities so that it can perform system enhancements during bottlenecks or server crashes.

1.3 Research Scoping

In this research, the researchers tried to apply the following limitations of the problem:

- The research only focuses on testing application performance with load testing and stress testing.
- Testing using black box testing without knowing the internal structure of the program.
- The API service tests were performed on an architecture consisting of the following server specifications; Processor: 4 Cores, RAM: 8Gb, Storage: 160Gb, OS : Ubuntu 18.04
- The application and database install in the same server.
- The test was performed using the open source JMetter tool which was run on the following hardware specifications; Processor: 2 Cores, RAM: 8Gb, Storage: SSD 128Gb, OS : Ubuntu 18.04.
- Test using a virtual user with the same request.
- The test targeted max 500 concurrent users.
- Optimization is done only in the configuration of the service application and web service.

2. Methodology

2.1 Performance Testing

Application testing consists of four elements: functional testing, compatibility testing, usability testing, and performance testing. If the software quality standard is compared to the ISO 25010 standard, the application should be tested for functionality, compatibility, usability, and performance.[2] Another definition of performance testing is a type of testing to ensure that software is functioning properly under the expected workload. The main goal is not to find bugs but to eliminate performance bottlenecks [3]. The focus of the performance test:

- Speed - determines whether the application responds fast enough.
- Scalability - determines whether the maximum number of user loads can be handled
- Stability - determines whether the application is stable with various loads.

When dispatch performance testing, there are 6 stages, namely the stages of project assessment, planning, scripting, test execution, results analysis, and reporting [4].

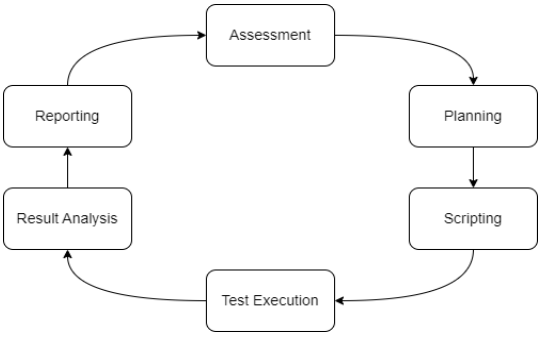


Figure 1 Performance Test as part Software Test [5]

2.2 Load and Stress Testing Methods

Method used to implement performance testing is load and stress testing. Load testing required to know the software solution will work under the real load, while stress test are needed to test stability and reliability of the system or it can be called as testing system durability [6].

Load testing is a performance testing technique that measures the response of a system under various load conditions. This test helps determine how the software behaves when multiple users are using the software at the same time. Load testing is required to simulate concurrent site/site usage [7]. Stress testing is a performance testing technique that uses virtual users who exceed the maximum number until system/application downtime occurs and is typically used for a longer period of time.

Blackbox testing is the functional testing of software without knowing the internal structure of the program. This test is important to do so that there are no errors in the flow of the program that has been created. Black Box Testing focuses on the functional specifications of the software, the set of input conditions and performs tests on the functional of the program [8].

2.3 Testing Tools

Apache JMeter can be used to test performance on both static and dynamic resources, dynamic web applications. [9] Dynamic is used as a testing tool because it has many features that can be used according to testing methods, including varied reporting so that it can make it easier to read and analyze the results of the test.

Application Performance Monitoring (APM) refers to managing the performance of a software application to ensure the expected level of service, as measured by performance metrics and user experience monitoring. The APM solution aims to detect and determine application performance issues before users are actually impacted by them [10].

The APM used as an application performance monitoring tool is Dynatrace. APM is needed to help analyze performance with several features that can

provide information on where and when the application runs smoothly or slowly. The selection of dynatrace as an APM is due to the fairly complete features where all resources can be controlled and the experience of testers using Dynatrace in monitoring performance and reporting that is easy to understand [11].

2.4 Unit Testing

Creating a test plan, what is done to divide the API Service in modules based on the main functions and roles of module? These modules will be tested on the test run. Below are the users who use the application by this case study; Authentication, Marketing Officer, Branch Manager and Checker.

Module division is based on the user or user of the application. The number of API endpoints in each module can be seen in the table on the number of modules to be tested.

Table 1. API module endpoint to be test.

No.	Test module	Number of API endpoints
1	Authentication	1
2	Marketing Officer	2
3	Branch Manager	2
4	Checker	1
Total number of endpoints to test		6

3. Results and Discussion

3.1 Testing Result

Based on test results of six endpoints with concurrent user 500, it was obtained as follows:

- API Login Testing Result

Based on the results of three login tests, a summary was obtained as follows:

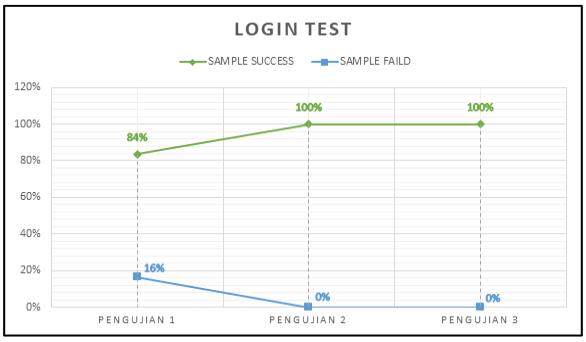


Figure 2 Testing Result based on success and failed access to API service.

As shown in Figure 2, this figure reveals information about the number of concurrent users who successfully accessed the API service function and who failed to access from the first try to the third try. Of course with some adjustments in the configuration file to achieve a 100% success level when accessing this function of API service.

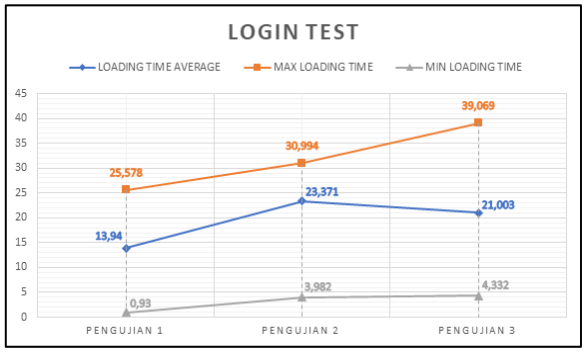


Figure 3 Testing Result based on min, max and average loading time access to API Service.

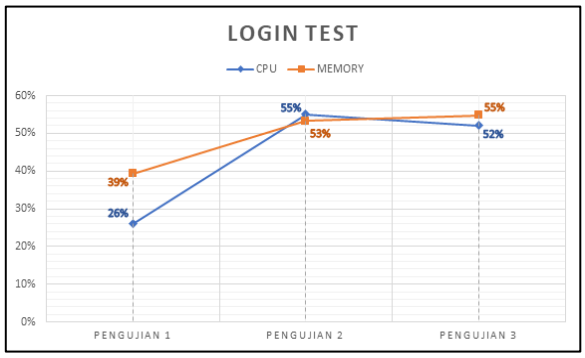


Figure 4 Testing Result based on CPU and RAM resource used to API Service.

As it can be seen in figure 3 and figure 4, average loading time, max loading time, min loading time, CPU and memory there is a fairly high comparison, this happen because the loading time is faster and the first test recorded 16% sample error which means there are 80 users out of a total of 500 users with a response below one seconds. Its affects results of loading time report.

Then in the figure 4 it shown CPU and memory usage is higher because in the first test only one service runs and the next test runs two services, which affects CPU and memory consumption, it's also found something catchy on second attempt that CPU utility higher than RAM utility, it's mean that CPU more busy than RAM, that's look good there no bottleneck on this process.

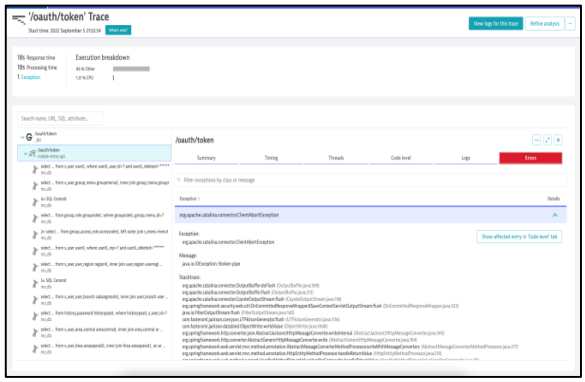


Figure 5 When the test to API Service failed it will show this error trace.

As shown in figure 5 the first test was failed and this show obtained sample, then tracing found an error on first request in the service log due to connection timeout between the service and the database. While here in figure 6 show there are 80 failed requests found in nginx web service log because service cannot respond to request.

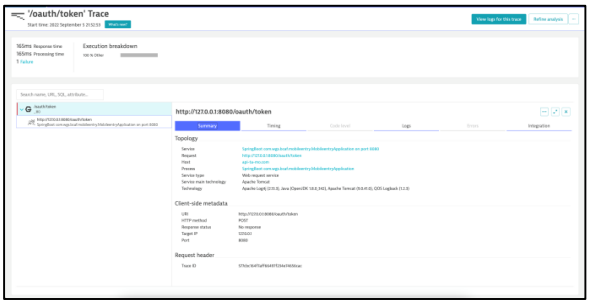


Figure 6 Testing Result to API Service failed will show log cannot respond request in nginx.

From the error found like show in figure 5 and figure 6, then researcher does some improvements or optimizations were made to the configuration of the nginx web service and application service, below are configuration code addition provided as follows:

1. Service configuration .

```
Server:
Port: 8080
Tomcat:
Threads:
max: 500
connection-timeout: 60s
```

2. Implementation of motode load balancing

```
upstream backend {
        server ip:8080;
        server ip:8081;
}
server {
        location / {
                proxy_pass: http://backend;
        }
}
```

After implementation of two technical suggestions on the second and third attempts, no request failed was found.

- Input Consumer API

Based on the results of three times testing save consumer data obtained a summary as follows.

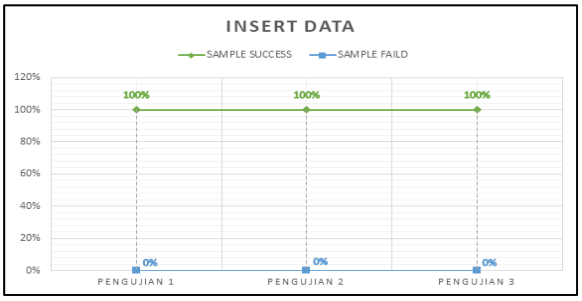


Figure 7 Testing results to insert data API service on success attempt.

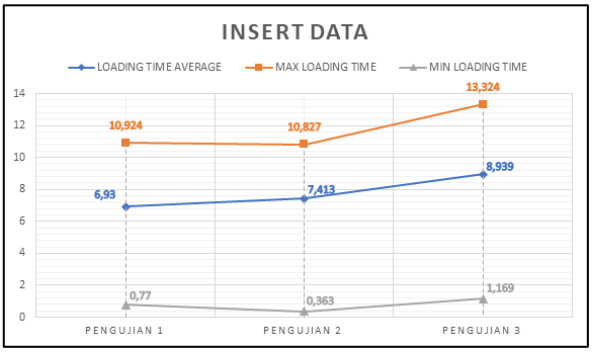


Figure 8 Testing Results to Insert Data API Service on min, max and average loading.

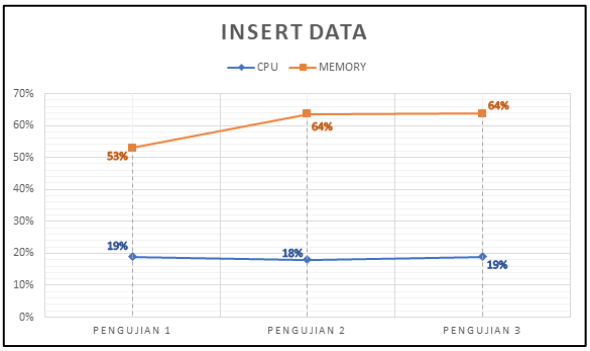


Figure 9 Testing Results to Insert Data API Service on CPU and RAM utility.

As seen on figure 7, figure 8 and figure 9, It can be seen in the average loading time, max loading time, min loading time, CPU and memory are quite stable, max loading time reaches 13 second due to the asynchronous process on the service waiting for the request that finishes processing first, but no request is failed.

As shown in Figure 9, in the first test, the CPU consumption rate was 19% and RAM was 53%, then in the second test, CPU was 18% and RAM was 64%, then in the third test, CPU was 19% and RAM was 64%, this indicates the insert function in API Service is stable.

- Update Data API BM

Based on the results of three update data to BM proceeding tests, the data obtained a summary as follows on figure 10.

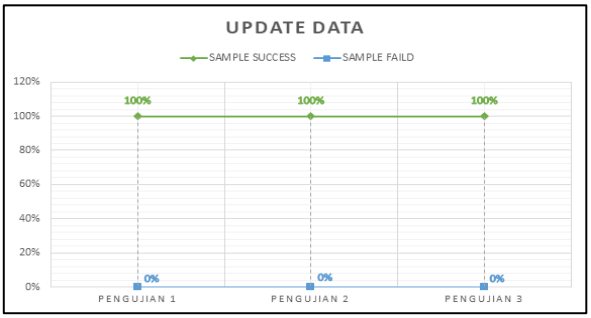


Figure 10 Testing results to update data API service on success attempt.

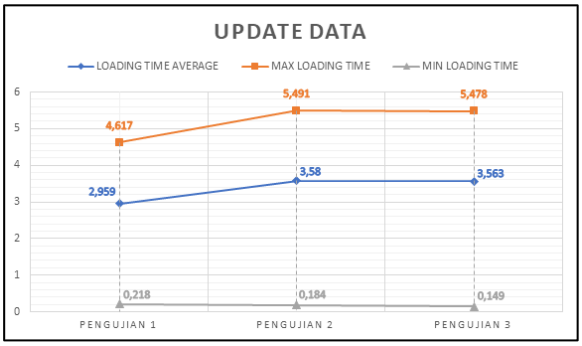


Figure 11 Testing results to update data API service on min, max and average load.

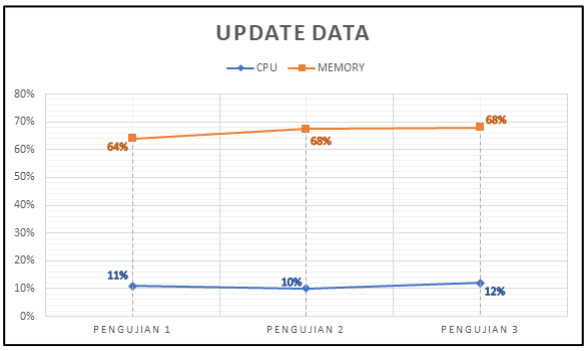


Figure 12 Testing results to update data API service on CPU and RAM utility.

As shown on figure 10, figure 11 and figure 12, It can be seen the average loading time between 2.959 to 3.58 second, max loading time between 4.617 to 5.491 second, and min loading time between 0.218 to 0.184 second, while in CPU utility reach between 10% to 12% of resource and memory between 64% to 68% of resource, this information show how quite stable on the test, max loading time reaches five second due to the asynchronous process on the service waiting for the request to finish processing, but no request failed.

- Update Data Approve API

Based on the results of three examination update data approve by BM, the data obtained a summary shown in figure 13, figure 14 and figure 15 as follows.

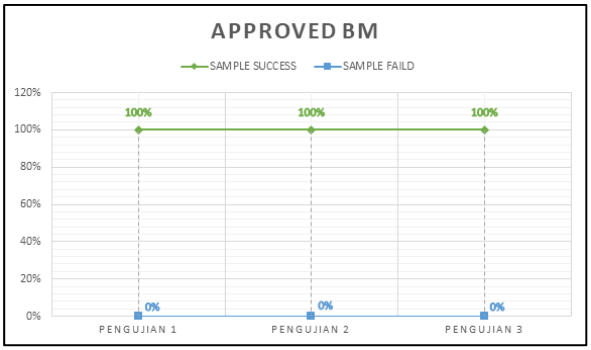


Figure 13 Testing results to update data approve API service on success attempt.

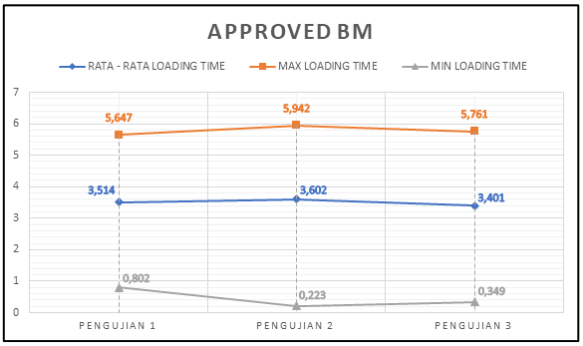


Figure 14 Testing results to update data approve API service on min, max and average load.

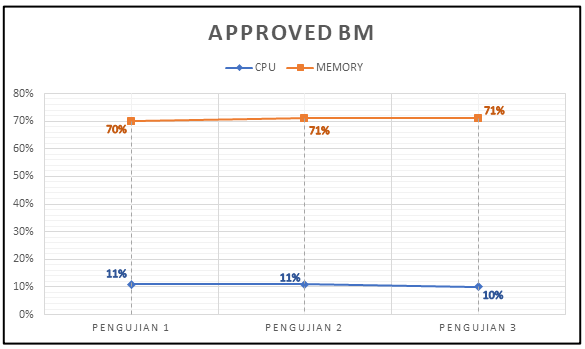


Figure 15 Testing results to update data approve API service on CPU and RAM utility

It shown that the average loading time, max loading time, min loading time, CPU and memory are quite stable, max loading time reaches average five second due to the asynchronous process on the service waiting for the request to finish processing, but no request failed.

- Update Data Reject API

Based on the results of three testing update data reject by BM, data obtained as shown in figure 16, figure 17 and figure 18 as follows.

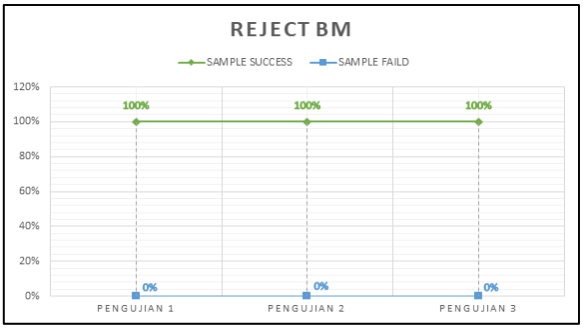


Figure 16 Testing results to update data reject API service on success attempt.

As happened in the previous API Service test, even in this test the observational data is still focused on the results of failed or successful tests, minimum, maximum and average loading as well as the percentage of CPU and RAM usage. With the results shown in Figure 16, Figure 17 and Figure 18 which shows a stable service function. It can be seen in the

average loading time, max loading time, min loading time, CPU and RAM are quite stable, max loading time reaches around five second due to the asynchronous process on the service waiting for the request to finish processing, but no request failed.

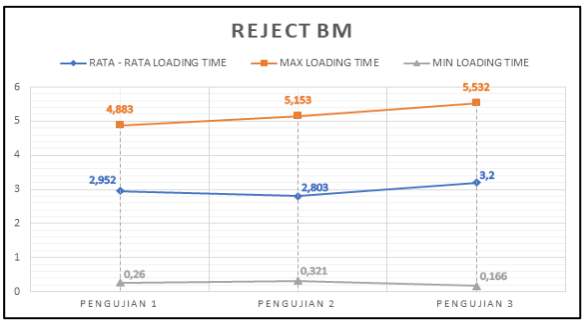


Figure 17 Testing results to update data reject API service on min, max and average load.

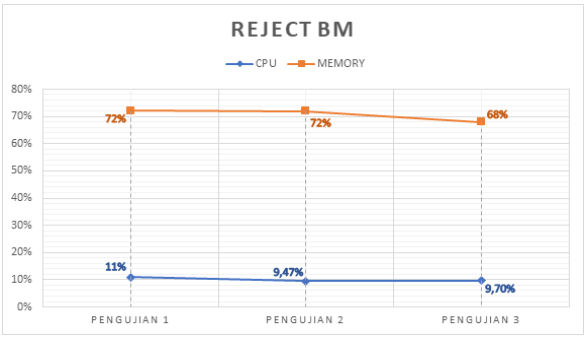


Figure 18 Testing results to update data reject API service on CPU and RAM utility

- Update Data Precede Checker API

The results of three times of precede checker testing, the data obtained a summary as follows.

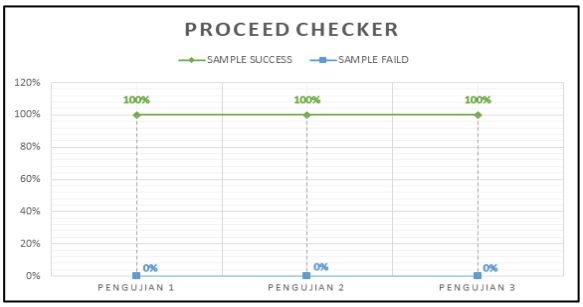


Figure 19 Testing results to update data precede API service on success attempt.

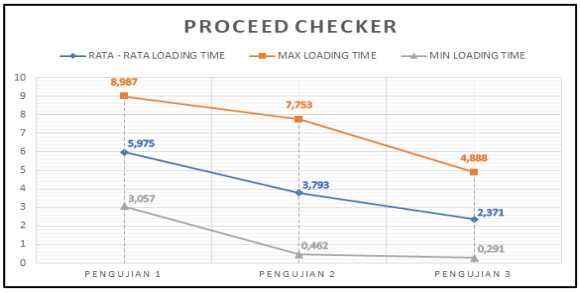


Figure 17 Testing results to update data precede API service on min, max and average load time.

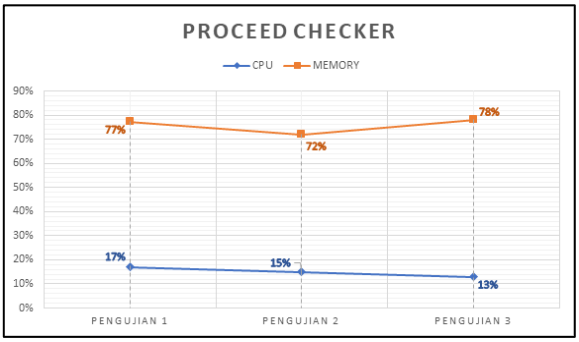


Figure 21 Testing results to update data precede checker API service on CPU and RAM utility

It seen in the average loading time, maximum loading time, minimum loading time, CPU and memory are quite stable, max loading time reaches eight second due to the *asynchronous* process on the service waited for the request that finishes processing, but no request are *failed*.

3.2 Technical Advice

Based on test results analysis in the first experiment on the login endpoint which was found 82 samples have failed, resulting in several technical suggestions being carried to improve performance, after implementation of technical advice in the next test did not get a failed response. Technical advice implemented as follows:

- Configure connection timeout on the web service application.

This suggestion is given from checked and analysis on Dynatrace results and found 81 samples contained in 1 request with a response of "500 internal server errors" on the application layer with this keywords error "ClientAbortException"

- Cause:
  The error is caused by a connection timeout in the web service application.
- Reference:
  https://stackoverflow.com/questions/43825908/org-apache-catalina-connector-clientabortexception-java-io-ioexception-apr-err
- Implementation:

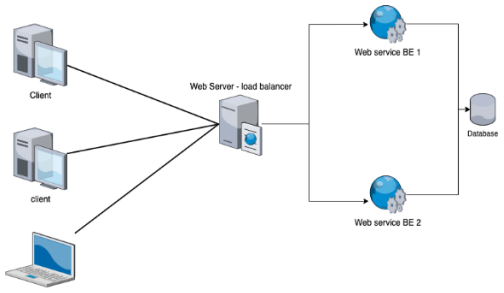
```
Server:
Port: 8080
Tomcat:
Threads:
max: 500
connection-timeout: 60s
```

- Impact:
  Response of some requests may take longer because the connection or processes are full.
- Network concept configuration with load balancing

This suggestion above given from Dynatrace results and found an error on the transport layer,

namely the proxy pass port 80 to the application port 8080 has an error because the application does not respond

- Cause: The error happen by possibility that the app is down or full of connection, so it can't be called.
- Solution: Distribute workloads across two or more network connections in a balanced manner so that work can run optimally and not overload on any of the connection paths.
  Load balancer is recommended for each service run on a different server or resource with optimal resource usage and harmonize by it.
  Here's the suggested topology:



- Implementation:

```
upstream backend {
    server ip:8080;
    server ip:8081;
}
server {
    location / {
       proxy_pass: http://backend;
    }
}
```

- Impact:
  Each workload is distributed across multiple *services*.

4. Conclusion

Based on research results that have been described in the previous path, it can be concluded that service with a running configuration cannot meet the expectations of requests with concurrent 500 users, so it is necessary to implement technical advice from analysis results in order to meet expectations and optimize performance. Server resources are still available when there a request with a concurrent of 500 users, but there are failed requests in several requests which mean it failed and not limited by resource specifications but requires optimization from the service side. The service does not show bottlenecks or downs when tested with 500 users concurrent request, for failed responses still occur at the beginning of the test because they have not implemented advance configuration. This is not because the service is unavailable so it requires someone to restart the service.

5. Acknowledgement

This research was supported by Langlangbuana University.

References

[1] Ginanjar, Arief, and Mokhamad Hendayun. "Spring framework reliability investigation against database bridging layer using Java platform." *Procedia Computer Science* 161 (2019): 1036-1045.

[2] David, Assaf Ben. "Mobile application testing (best practices to ensure quality)." (2011).

[3] Weyuker, Elaine J., and Filippos I. Vokolos. "Experience with performance testing of software systems: issues, an approach, and case study." *IEEE transactions on software engineering* 26.12 (2000): 1147-1156.

[4] Du Plessis, Marina. "The role of knowledge management in innovation." *Journal of knowledge management* (2007).

[5] Vokolos, Filippos I., and Elaine J. Weyuker. "Performance testing of software systems." *Proceedings of the 1st International Workshop on Software and Performance*. 1998..

[6] Erinle, Bayo. *Performance Testing with JMeter 3*. Packt Publishing Ltd, 2017.

[7] Permatasari, Desy Intan. "Pengujian aplikasi menggunakan metode load testing dengan apache jmeter pada sistem informasi pertanian." *JUSTIN (Jurnal Sistem dan Teknologi Informasi)* 8.1 (2020): 135-139..

[8] Abbas, Rabiya, Zainab Sultan, and Shahid Nazir Bhatti. "Comparative analysis of automated load testing tools: Apache jmeter, microsoft visual studio (tfs), loadrunner, siege." *2017 international conference on communication technologies (comtech)*. IEEE, 2017.

[10] Rosa, Ariani Sukamto. "Rekayasa perangkat lunak terstruktur dan berorientasi objek." (2016).

[11] Willnecker, Felix, et al. "Using dynatrace monitoring data for generating performance models of java ee applications." *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*. 2015.