ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 8, August 2016

Performance Enhancement of Fare Collection System

Sharada Srinivas

P.G. Student, Department of Information Science Engineering, M S Ramaiah Institute of Technology, Bangalore, Karnataka, India

ABSTRACT: It is a matter of great concern that many Software Engineers consider design, development and deployment of Software products or solutions to be easily achievable. Over simplifying the Software Engineering activities as per the SDLC has resulted in Billions of Dollars being wasted on Software solutions that has not met the Scalability and Performance requirements when deployed. The major challenge for organizations is to ensure that their technology systems are efficient, scalable and sustainable and improve business performance at a low cost of ownership. Over its lifetime, the cost of a software product is determined by how well it achieves its objectives for quality attributes such as performance, reliability/availability or maintainability. There are not many engineers who consider developing solution from a performance stand point. Software solution not meeting the required performance objectives is often considered as the biggest risk to the success of a project. Reducing this risk will make a major contribution to reducing the total risk to the project overall. It is very crucial for every software engineer to realise the importance of performance in developing a software solution. Higher the performance, higher the value of the product. Here in this Project, I provide a case where in the software solution was developed without considering the performance requirements in the beginning. Later, realising that the product was short of the required performance level, tuning had to be done without affecting the design of the working solution, which posed to be the biggest challenge.

KEYWORDS: Software Performance, Apache Http Server, Apache Tomcat, MySQL NDB Cluster

I. INTRODUCTION

The case study provides an example of proprietary fare collection software solution which was designed without prior knowledge regarding the required performance and no measurements done to qualify the solution. As shown in Fig.1, according to the initial architecture, Electronic ticket machines (ETMs) are used to send ticket transaction data. A load balancer forwards the transaction data from ETMs to an application server. Here two application servers have been used to achieve fail over mechanism. All the transactions pass through web application servers which in turn send the transaction data to the database server.

The solution was successfully developed to achieve the intended functionality and designed to handle upto 300 concurrent users only. But soon the organization realised that it did not meet the required performance standards and lacked scalability to meet the market needs. When spoken to management, they were very clear that they wanted to scale the solution from 300 to 1000 users and tune the solution to achieve performance to a best acceptable level without discarding the complete working solution as well as not triggering major design changes.

Now there was a need to come up with a complete plan of action regarding how we were going to achieve the required goals. A small plan was created consisting of the following activities:

1. Perform tests to measure the solution and analyse the current solution
2. Identify the Performance bottlenecks
3. Performance Enhancement – Propose changes to improve the solution
4. Retest the solution to measure to what extent has the performance improved

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 8, August 2016

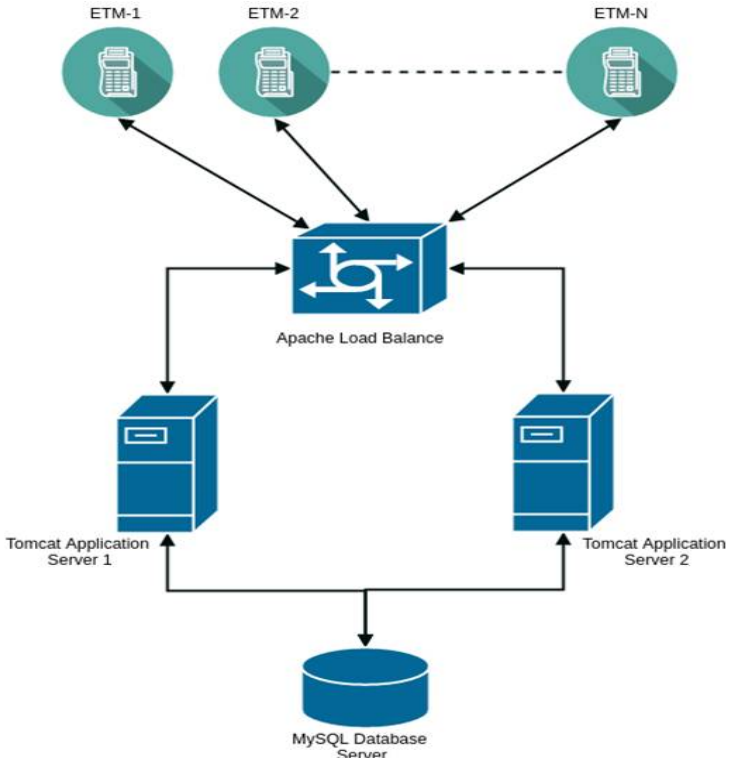

Fig.1. Fare Collection System Initial Architecture

II. RELATED WORK

Application performance can be determined greatly by the experience of the end users.Generally, when assessing the application performance, the following information is required:

1. Number of users who use the application
2. Number of users who concurrently use the application
3. Number of additional users the application can take over time (Scalability)

Thereare various factors that affect application performance:

1. Architecture and design of the application
2. Application code developed (complexity and functionality)
3. Server configuration
4. Database design and database server configuration settings
5. Middle-ware components
6. Workload during peak usage (user population)
7. Resource consumption (memory/hardware/processor/network)

Main criterion to be measured to know the performance of the application are as follows:

1. Availability: The amount of time the application is operable as per its specification. Usually measured in terms of uptime or downtime of the application.
2. Response time: The amount of time the application takes to respond to a user request. Ex-transaction per second, requests per second, etc.
3. Throughput: Rate at which application oriented events occur. Ex- number of hits on a web page in a given period of time.
4. Concurrency: Number of users simultaneously using the application.

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 8, August 2016

5. Utilization: Amount of various resources the application uses. Ex- network utilization measure such as data volume, data throughput, data error rate, etc. server utilization measures such as CPU utilisation, memory utilization, disk input/output, etc.

III. ANALYSE CURRENT SOLUTION

To evaluate the current performance of the system, first step was to collect the required performance metrics and second was to analyse the results to draw conclusion regarding the performance status of the solution. To collect the required metrics, we did performance tests using JMeter tool.

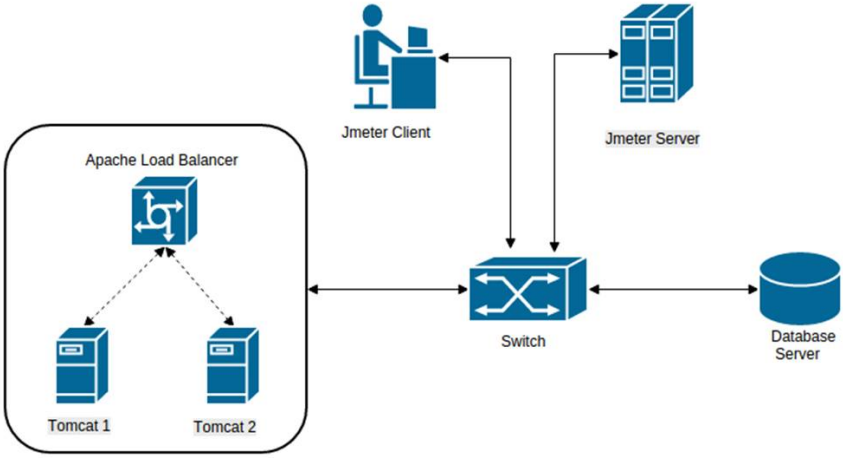

Fig.2. Performance Test Set up 1

Performance test set up environment was as shown in Fig.2. We made use of PCs that were set up as load balancer, application and database servers.
The application servers were of the below configuration,

RAM	: 3 GB
Architecture	: i686
CPUs	: 2
CPU MHz	: 2800
Cache	: 2 MB

Database server was of the below configuration,

RAM	: 8 GB
Architecture	: x86*64
CPU MHz	: 6948
Cache	:3 MB

Apache HTTP server was set up on one of the application servers that acted as a load balancer, balancing requests between two Tomcat Application Servers.

1. Apache Load Balancer Configuration: Apache 2.4 HTTP server was set up as Load Balancer. mod_jk module used to configure Apache as load balancer to balance the load between both the application servers. Load balancing method configured was to balance load based on "R"equest – R method. lb factor which decides the load that needs to be transferred to different instances of tomcat was set to 1, indicating equal balancing of load among the tomcat instances.
2. Tomcat Configuration: Tomcat 7 was set up on both the application servers. Tomcat mainly consists of two configurations:

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 8, August 2016

a. Connector configuration – AJP connector here in this case. AJP connector is used on tomcat 7 for communication of tomcat with Apache HTTP server, if mod_jk module is used for load balancing. Below is the configuration of some of the main parameters that was done,
protocol=org.apache.coyote.ajp.AjpNioProtocol (AJP NIO Protocol)
connectionTimeout=1000
maxSpareThreads=3000
maxThreads=3000
minSpareThreads=3000
bufferSize=1024

b. JDBC Pool Configuration - JDBC Pool is a pool of connections that are kept by Tomcat for instant communication with MySQL. There are various settings that needs to be done such as providing application specific data source name using which the client connections can connect to the database, database schema name as well as connection parameters such as username and password, and several other thread related parameters. Below is some of the important JDBC pool configuration that was used.
useLocalSessionState=true
defaultAutoCommit=false
maxActive=300
maxIdle=200
minIdle= 100
maxWait=3000
defaultTransactionIsolation=READ_COMMITTED

3. MySQL Database configuration: MySQL configuration parameters such as concurrent threads, query time-out, wait time-out, connection time-out, etc were all kept at its default. InnoDB engine was used and InnoDB related parameters were also kept to their default values.

IV. INITIAL PERFORMANCE TESTS

The scenario that need testing can be depicted as shown below in Fig.3. Each ETM device issues tickets whose transaction data are sent to the ETM Application server which in turn parses the transaction data obtained and insert required data in to the database.

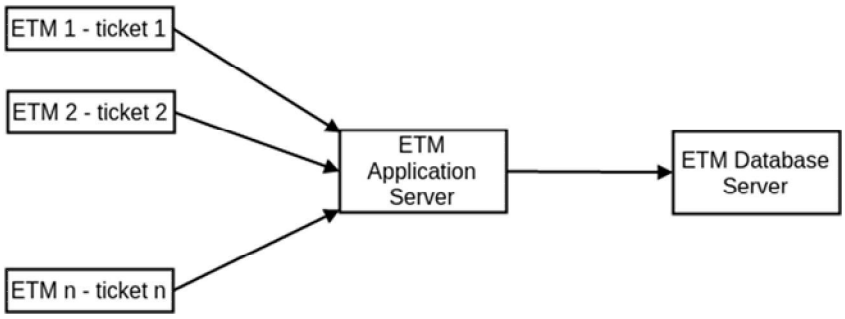

Fig.3. Testing Scenario

We want to test the number of concurrent requests from ETMs that our system can support i.e., testing for number of concurrent tickets that can be inserted in DB.
We started tests from 200 concurrent ETMs, by increasing 50 for each iteration until we obtained DB error in the JMeter which depicts failure of insertion of ticket information in the DB. According to our tests done we were able to achieve just 300 concurrency as the test failed for 350 users with DB write error of 15.9%.

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 8, August 2016

Below are the testing results for 300as well as 350 users(Table.1.). According to the test results, it was clear that the tests passed for 300 users without any error but failed for 350 users with 15.71% error in writing the transaction data to the database.

Number of Users	Response time in ms	Min Response time in ms	Max Response time in ms	Connect time in ms	DB write error %
300	350	83	743	2.88	0
350	392.67	79	745	2.24	15.71

Table.1. Initial Performance Test Results

V. IDENTIFY PERFORMANCE BOTTLENECKS

According to our tests done, it was clear that with application server and database server set up with InnoDB engine that we had would support us with only 300 concurrent ETM users.
As observed the solution had two major drawbacks

1. Solution was not scalable. The database used MySQL Inno DB Engine which does not support scalability. It also does not provide for data high availability. From the above architecture, it was clear that there is a single point of failure. If the database server goes down, then whole solution comes to a halt. One of the major enhancementthat was to be made was to remove this single point of failure.
2. Tomcat configurations were set to its minimal configuration. Though in many cases these configuration is enough to obtain a certain level of performance, to gain better one must make sure that the configuration settings are set to the required optimised definite value.

VI. PERFORMANCE ENHANCEMENT

1. Tomcat Performance Enhancement: Tomcat Performance Enhancement settings was done mainly at two places – one at JDBC pool configuration and second at Tomcat AJP configuration.Various parameters were added to the JDBC Pool configuration which made us control our threads pool more accurately. The following parameters were added.
    a. initialSize: This parameter defines the number of initial threads that must be started in the pool on tomcat start up.
    b. testWhileIdle: There are chances of threads that are in the pool to become stale with loosing connection from DB. To avoid these stale connections, we can validate the threads regularly by setting this parameter to TRUE.
    c. validationQuery: This is a simple query such as SELECT 1 that can be used to validate the threads as and when required.
    d. validationInterval: Interval in milliseconds for which threads are validated. This avoids excess validation.
    e. timeBetweenEvictionRunsMillis: The number of milliseconds to sleep between runs of the idle connection validation/cleaner thread. It dictates how often we check for idle, abandoned connections, and how often we validate idle connections.

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 8, August 2016

f. removeAbandoned: The stale threads if any will be abandoned that is removed from the pool if this parameter is set to TRUE. Also sometimes client threads may abandon the pool connection unexpectedly. This parameter can also be used to remover such threads from the pool.
g. RemoveAbandonedTimeout: Regular interval in milliseconds at which the JDBC pool can be check for stale connections and abandoned them.

Our JDBC Pool Configuration was modified as follows:
useLocalSessionState=true
defaultAutoCommit=false
validationQuery=SELECT 1
removeAbandoned=true
removeAbandonedTimeout=60
maxActive=1000
maxIdle=500
minIdle= 100
initialSize=100
maxWait=3000
testWhileIdle=true
timeBetweenEvictionRunsMillis=140000
validationInterval = 34000
minEvictableIdleTimeMillis = 150000
defaultTransactionIsolation=READ_COMMITTED
Based on the above JDBC pool settings our Tomcat AJP connector settings was also modified so that we could get that the equal number of connections from Apache. The AJP connector configuration was modified as follows:
maxThreads=1000
minSpareThreads=500

2. MySQL Performance Enhancement: Main change that was made to get better performance was change MySQL Engine from InnoDB to NDB. Two data nodes were set up so that we could eliminate single point of failure. We also obtained scalability as we can add data nodes as and when the load and data become more as and when required. Also data in one data node is also available in the secondary sections of the other data nodes. If any data node fails, the same data can be obtained from the secondary portions of the other data nodes. Hence this also provided for high availability.

VII. EXPERIMENTAL RESULTS

Performance tests were conducted again similar to the manner done previously to evaluate the performance of the modified solution. Fig.4 depicts the test set up that was done, where we made use of PCs like that was done previously to set up load balancer, application and database servers.
The Apache HTTP server was of the below configuration,
RAM : 3 GB
Architecture : i686
CPUs : 2
CPU MHz : 2800
Cache : 2 MB
The application servers were of the below configuration,
RAM : 3 GB
Architecture : i686
CPUs : 2
CPU MHz : 2800
Cache : 2 MB

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 8, August 2016

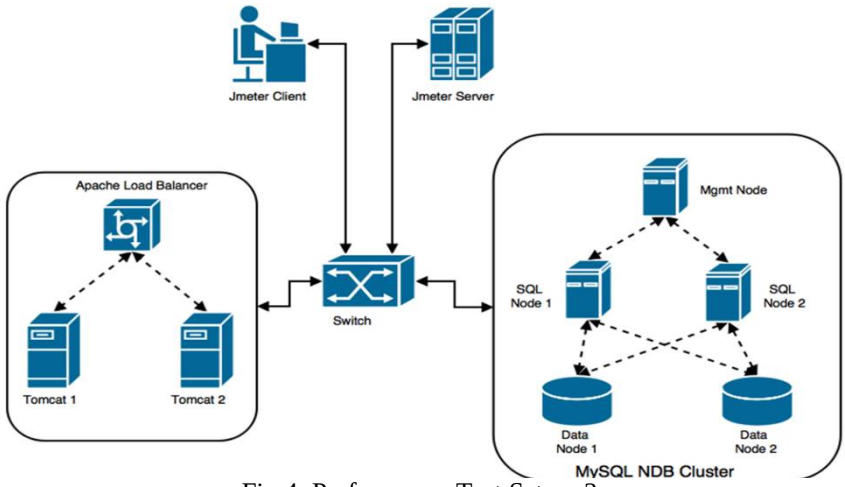

Fig.4. Performance Test Set up 2

NDB data nodes were of the below configuration,
RAM : 8 GB
Architecture : x86*64
CPU MHz : 6948
Cache :3 MB
Apache HTTP server that acted as a load balancer, balancing requests between two Tomcat Application Servers were set up along with NDB Management node on the same PC. 2 separate Tomcat application servers were set up on two separate PCs along with one NDB SQL node each. Two separate PCs were used as two separate NDB data nodes.
We started tests from 200 concurrent ETMs, by increasing certain number of requests for each iteration until we obtained until we reached 1000 user concurrency or obtained DB error in the JMeter which depicts failure of insertion of ticket information in the DB. According to our tests done we were able to achieve 1000 concurrency without any DB write error.
Table.2. shows the results that were obtained for 1000 users. The average response time for 1000 users obtained was 311ms without any error in writing to Database.

Number of Users	Response time in ms	Min Response time in ms	Max Response time in ms	Connect time in ms	DB write error %
1000	311	28	1717	104.499	0

Table. 2. Performance test Results 2

According to our tests done, it was clear that with application server and database server set up with NDB Cluster engine that we were able to achieve the required concurrency. We also had achieved data high availability as well as scalability by the use of NDB Cluster. We also had eliminated the single point of failure. It is clear from the analysis done that our new modified set up was far better than the original one as we had achieved all the required objectives that were intended.

VIII. CONCLUSION

Despite knowing the importance of performance, various organizations fail to give heed to performance requirements of a software solution. By realising the importance of software performance not only can the organization mitigate the

ISSN(Online): 2319-8753
ISSN (Print): 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Issue 8, August 2016

performance risks but also can obtain greater ROI and higher user acceptance. At the end of the project completion, the following was achieved:

1. Improved overall performance of the software application.
2. Higher ROI as the software adoption rates became higher once the performance of the product was guaranteed.
3. Higher quality of the applications.
4. By the increase in the performance of the product, decrease in degradation of service quality over time.

REFERENCES

[1] Ian Molyneaux, "The Art of Application Performance Testing, learn how to test web applications using Apache JMeter with practical, hands-on examples", First Edition, January 2009
[2] Bayo Ernile , "Performance Testing with Jmeter 2.9" , July 2013
[3] Apache Software Foundation, "Apache JMeter Documentation", May 9, 2016
[4] Apache Software Foundation, "Apache HTTP Server Documentation Version 2.4", April 10, 2016
[5] Apache Software Foundation, "Apache Tomcat 7, Version 7.0.70", Jun 15 2016
[6] Alex Davies, "High Availability MySQL cookbook", April 2010
[7] Charles Bell, Mats Kindahl, Lars Thalmann, "Mysql High Availability, Tools for building Robust Data Centers", 2nd edition, April 2014
[8] Henry H. Liu, "Software Performance and Scalability, A Quantitative Approach", 2009
[9] Bayo Erinle , "JMeter Cookbook, 70 insightful and practical recipes to help you successfully use Apache JMeter", October 2014
[10] Connie U. Smith, Lloyd G. Williams, "Best Practices for Software Performance Engineering", Performance Engineering Services and Software Engineering Research, 2003
[11] André B. Bondi, "Foundations of Software and System Performance Engineering, Process, Performance Modeling, Requirements, Testing, Scalability, and Practice", August 2014
[12] Lloyd G. Williams, Connie U. Smith ,"The Business Case for Software Performance Engineering", Performance Engineering Services and Software Engineering Research, March 2002