**IMPROVING PERFORMANCE AND SECURITY IN BYZANTINE FAULT TOLERANT** **SYSTEMS**

A Project Report

CECS 546 - Fault Tolerant Computing Systems

California State University, Long Beach

By Sarveshwaran Sampathkumar

Department of Computer Science

California State University Long Beach

Long Beach, United States of America

sarveshswaran@gmail.com

ABSTRACT

**IMPROVING PERFORMANCE AND SECURITY IN BYZANTINE FAULT TOLERANT** **SYSTEMS**

By

Sarveshwaran Sampathkumar

July 2018

In a fault tolerant computer system, Byzantine Fault Tolerant (BFT) is where the components fail or produces incorrect results or the inability of the component to convey if the component has failed to the requestor. These types of failures are very general in nature and they are very difficult to detect by the failure detection system. The existing Byzantine Fault Tolerance systems tries to solve the Byzantine failures by having a single leader system which serves all the requests from the clients. The issue with this system is that it possesses a risk of single point of failure. Moreover, having a single leader server is less secure. Our proposal to solve these issues is to have multiple replicas of the leader server to serve the client requests. Having multiple server helps in distributing the load evenly across the servers and improves the performance of the Byzantine Fault Tolerant systems. To improve the security, we implement encryption and decryption techniques during the data transfer between the clients and servers. This ensures that the data transfer is safe, and the data is unaltered by external forces like hackers etc.

**TABLE OF CONTENTS**

ABSTRACT………………………………………………………………………………... i

LIST OF FIGURES…………………………………………………………………...…… iii

1. INTRODUCTION…………………………………………………………………. 1
2. BYZANTINE FAULT TOLERANT SYSTEM…………………………………… 2
3. EXISTING SYSTEM……………………………………………………………… 3
4. PROPOSED SYSTEM – SAREK…………………………………………………. 4
5. CRYPTOGRAPHIC SERVICE…………………………………………………… 7
6. IMPLEMENTATION AND RESULTS…………………………………................ 8
7. FUTURE IMPROVEMENTS………………………………………………………13
8. CONCLUSION…………………………………………………………………...... 14

REFERENCES………………………………………………………………….................. 15

**LIST OF FIGURES**

1. SAREK System Architecture ……………………………………………………… 4
2. Algorithm – Predict and Order Requests ………………………………………….. 6
3. Algorithm – RSA ..……………………………..………………………………….. 7
4. Homepage of the application ……………………………………………………… 8
5. Database Table …………………………………………………………………….. 9
6. Table view to the user ……………………………………………………………..10
7. Table stored as text file ……………………………………………………………10
8. Elapsed time for the scenarios …………………………………………………….11
9. Graph for the scenarios ………………………………………………....…………11
10. Output comparison for all the scenarios …………………………………………..12

**CHAPTER 1**

**INTRODUCTION**

In the current era of internet, the demand for information from the users have grown at an exponential rate. It is very important for the content providers to provide correct results to the requesting clients. Providing incorrect results to the clients leads to loss of clients due to the heavy competition provided by the competitors. Though the content provider can equip their system with latest and fastest technologies, they cannot always guarantee providing correct results to the users. This is due to the presence of external factors which affect the servers from providing the correct results to the clients. One such reason is the Byzantine issue. The Byzantine issue is when the system is incapable of informing if they are working as expected or have been affected due to other factors resulting in providing incorrect results to the clients. This issue is very common in the field of computers and hence it is very important for the service providers to develop a system which is fault tolerant to Byzantine issue. In this project, we will see in detail about the Byzantine issue in the computers, how it is caused, what is the existing solution for this issue and what is the new system which we are implementing to improve the performance and security in the Byzantine Fault Tolerant system. Apart from these, we will also look into the cryptographic algorithm which will be used in the Byzantine Fault Tolerance systems to improve the security.

**CHAPTER 2**

**BYZANTINE FAULT TOLERANT SYSTEM**

To understand the Byzantine problem in general, lets take an example of Byzantine army who are trying to capture an enemy fort. The Byzantine army has been divided into multiple divisions and each division has a commander. The constrains over here is that, the commanders can communicate with each other only through a messenger. In order for the Byzantine army to capture the fort, all the commanders should take the same decision, that is to either attack or retreat. Few of the commanders may prefer to attack where few may prefer to retreat. The commanders in the Byzantine army could be a faithful commander or a traitor. The general idea in this problem is that it is difficult to find out which of the commanders are traitors and which are not, all the commanders should have a common algorithm to make sure that everyone takes the same decision and also that the traitors cannot be hinderance in acquiring the enemy fort.

The above general Byzantine Issue can be related to the world of computers as follows. The commanders here are the servers who serve the clients and the messenger who sends the message from one commander to another can be compared to as communication links between the servers and the clients. To solve this issue, we get the response from all the servers and find the result which was provided by most of the servers. By doing so we get to choose the right value as calculated my majority of the servers. We could define a Byzantine fault as the one which shows different signs for different machines. And a Byzantine Failure is defined as a failure which is caused due to the Byzantine Fault. These Byzantine faults can be due to external factors such as hackers, electrical circuit failures etc. Hence it is very important to design a system which is tolerant to the Byzantine Failures. Such systems are called Byzantine Fault Tolerant systems.

**CHAPTER 3**

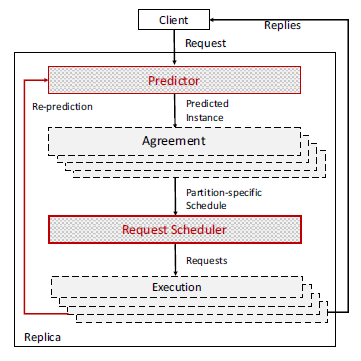
**EXISTING SYSTEM**

In the existing system of Byzantine fault tolerant system, we consider the below client server system to explain it. There are multiple clients trying to access information from the server. In the existing Byzantine Fault Tolerance system, the requests from the clients are processed in a sequential order. In the existing system, there lies only one leader server. The client’s requests are processed by the leader server. The job of the leader server is to send the user requests to all the main servers which will be processing these requests from the clients. All the main servers process the client request and provide the result back to the leader server. The leader server, on receiving the results from all the main servers, compares their results with all others. The result which was produced by majority of the servers are considered as the correct result by the leader server. The Leader server, then requests one the main server which has produced the correct result to respond to the user. By doing so, the results produced by the affected servers will never be served to the users. Apart from choosing the correct result from the server, the leader server also sends a refresh request to that main server, in order to refresh the data on that server with the correct data. The disadvantage with the current system is that it has single leader. Having a single leader increases lot of over head on that server due to the fact that the number of requests received from the client is humongous. This in turn limits the performance and the scalability of the Byzantine Fault Tolerant system. Moreover, this system is susceptible to single point of failure. In case of the failure of the leader, the compete system would be down creating a bigger impact to the system.

**CHAPTER 4**

**PROPOSED SYSTEM – SAREK**

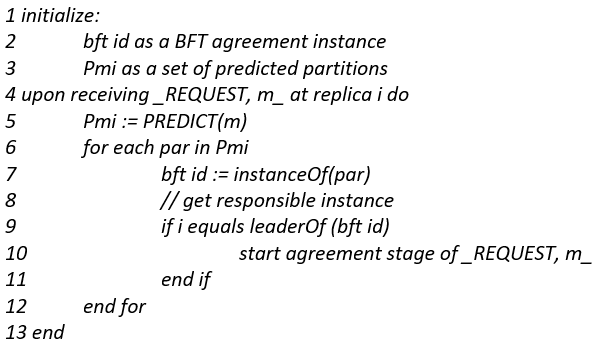
In order to solve the issues faced by the current Byzantine Fault Tolerant system, in this project we introduce a new concept called SAREK [1]. SAREK enables parallel processing during both the stages of the processing – agreement and execution. In this system, unlike the existing system where we have only one leader, we have multiple leaders. By this way SAREK enhances the performance of the system by promoting both parallelism and also by balancing the load across the main servers. The proposed system makes use of functions such as PREDICT and WATCHDOG which are explained in the below paragraphs. The architecture of SAREK is shown below. We can see that the agreement and execution are separated from each other.



**FIGURE 1.** **SAREK System Architecture.**

The SARAK system is similar to the existing system but with the exception that in this system we have multiple leader rather than having single leader as in case of the existing Byzantine Fault Tolerant system. There are multiple clients trying to access information from the server. In this system, all the main servers are interconnected with each other. Whenever a client requests for some information, the server which receives the request computes the result for the client while simultaneously sending the same request to all the other connected main servers. All the other servers also compute for the same request requested by the client and responds to the main server. The main server receives all the responses from the server and finds the answer which has been reported by maximum of the servers. It then compares this result with the result computed by itself. If both the results are the same, then the main server itself responds back to the requested client. If in case the result computed by the main server is different from the one which has been computed from all other main servers, then this server refreshes its data with any one of the main server which has computed the correct result. Also, it requests any one of the main server which has computed the correct result to respond back to the client. One significant rule followed here is that it doesn’t direct the same server from which it is refreshing the data to respond to the client. Instead, it directs one of the many servers with the correct data to respond to the client. By doing to, the time taken for the processing time for the request is reduced. Moreover, the server also updates it data parallelly while processing the clients request.

In the proposed system, all the main server acts as a leader. This enables parallelism during both agreement and execution stage. Every server is equipped with functions – PREDICT and WATCHDOG. The predict function sends the user requests to all other main servers which are connected with that server. The algorithm for predict and order requests is as follows:



**FIGURE 2. Algorithm – Predict and Order Requests.**

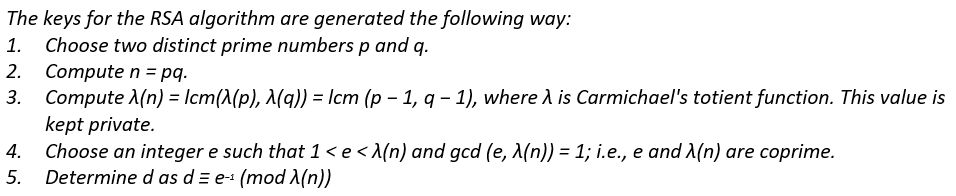
The WATCHDOG function computes the correct answer by comparing all the results returned by all the connected main servers. It compares all the values and returns a value which was reported by majority of the main servers.

**CHAPTER 5**

**CRYPTOGRAPHIC SERVICE**

Now that the SAREK system has been implemented, we could guarantee that the result returned by the system would be Byzantine fault free. A system can never be termed good until the user gets the correct data in return for the request sent. SAREK Byzantine Fault Tolerant system ensures that the result returned by the system is correct. But it doesn’t guarantee the data received by the user. This is because SAREK Byzantine Fault Tolerant system cannot be responsible for the data transfer from the system to the user. Hence this is still open to security concerns.

In order to solve the above-mentioned issue, we make use of the cryptographic algorithm called RSA. RSA algorithm is also called as asymmetric cryptographic algorithm [2] which encrypts and decrypts the data using the public and private key. This ensures that the data transfer between the server and client is secure. Implementing this to the SAREK Byzantine Fault Tolerant system makes this a complete system which is very secure and also improvised in performance when compared to the existing Byzantine Fault Tolerant systems. RSA algorithm which is implemented in this project is as follows [3] [4]:



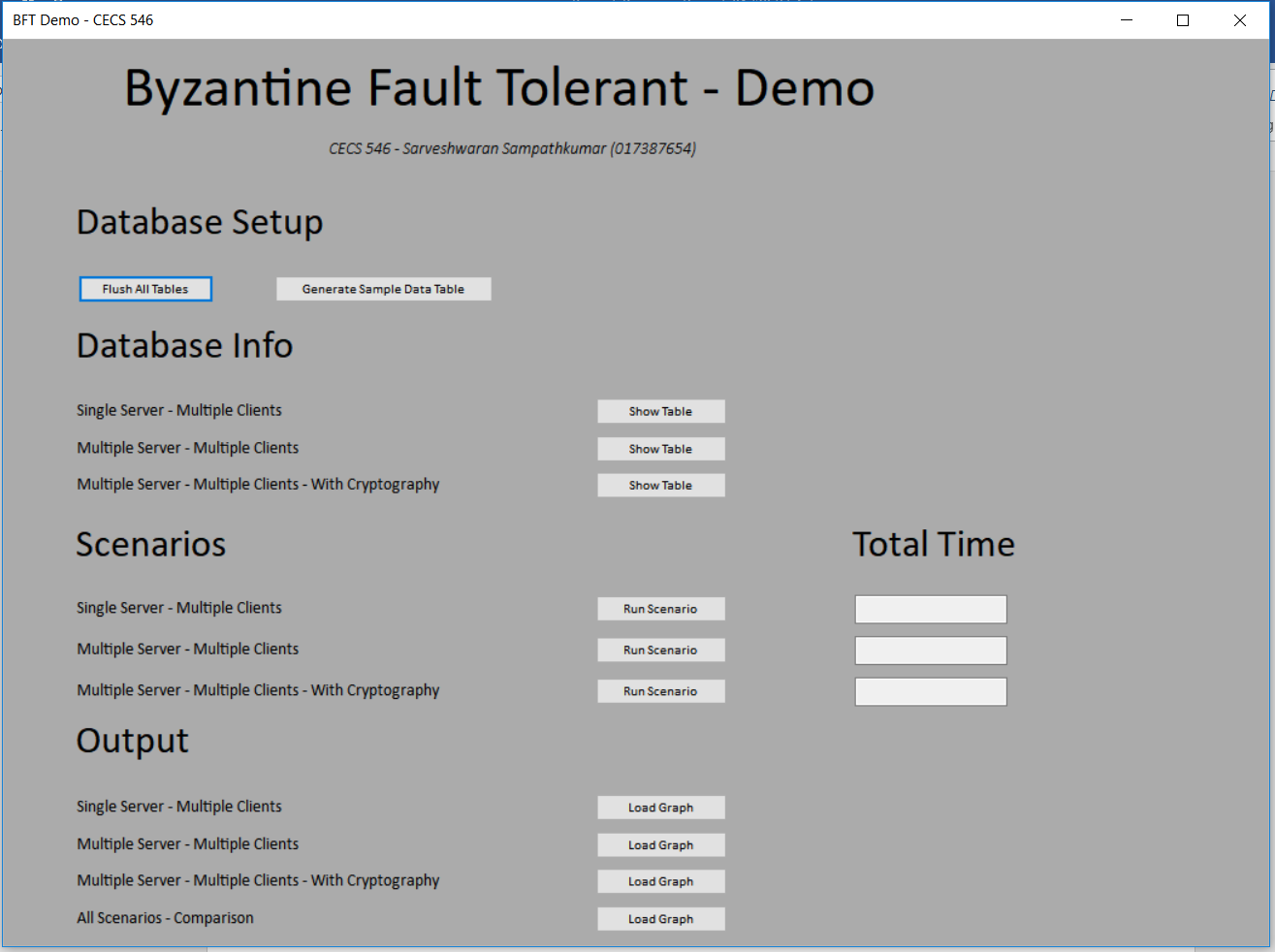
**FIGURE 3. Algorithm – RSA.**

**CHAPTER 6**

**IMPLEMENTATION AND RESULTS**

In order to implement the above discussed existing Byzantine Fault Tolerant system and SAREK Byzantine Fault Tolerant system, we make use of tools such as Microsoft Visual Studio and MySQL. Visual Studio and MySQL was installed on a system which is equipped with Intel i7 processor, 12GB of RAM and 1 TB of storage. Visual Studio was used to simulate different scenarios of client server system and MySQL was used to store the data generated from these scenarios on the system database. 1000 client requests are simulated in all the scenarios. The home interface of the application is shown below. It has 4 major layouts which are mentioned below:

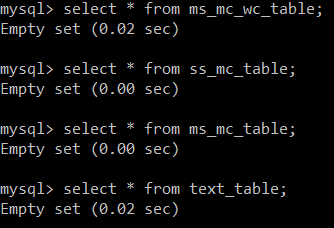
1. Database Setup – Sets up the database table before we start with the simulations.
2. Database Info – Shows the table to the user. Also stores the data in a file.
3. Scenarios – Different scenarios are performed in this layout.
4. Output – The output is shown as graph to the user in this layout.



**FIGURE 4. Homepage of the application.**

**Database Setup**

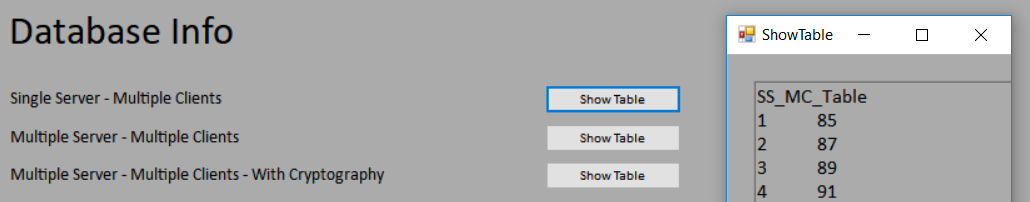
It has multiple buttons to perform various different operations. The first operation which has to be performed is to clear all the data in all the tables. This operation has been configured on the Flush All Tables button. Clicking on that button deletes all the data from the database. The next operation is to generate the number of user requests which would be made for the simulation. This has been configured on the Generate Sample Data Table button.



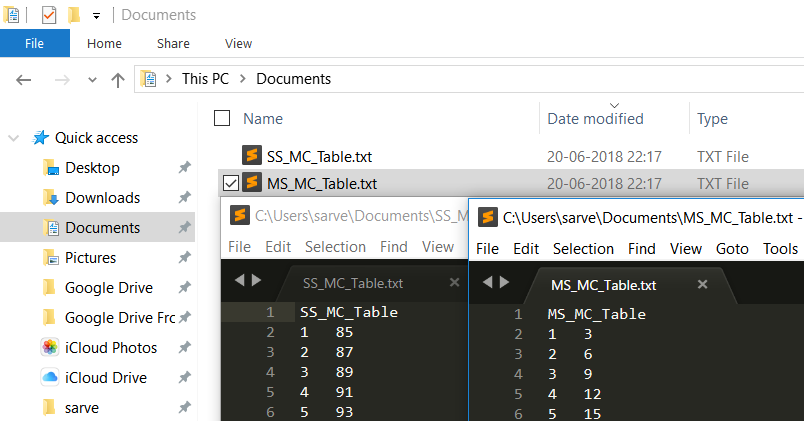
**FIGURE 5. Shows the database table.**

**Database Info**

This layout has multiple buttons to show the tables to the user for their respective scenarios. Clicking on the button also saves the table in a text file in the system’s My Documents directory. Usually these buttons are clicked once we perform the scenarios in the next section.



**FIGURE 6. Shows how the table is shown to the user.**



**FIGURE 7. Shows the table being stored as a text file.**

**Scenarios**

Three different scenarios are implemented in this project. They are as follows:

1. Single Server – Multiple Clients
2. Multiple Server – Multiple Clients
3. Multiple Server – Multiple Clients with Cryptographic Service

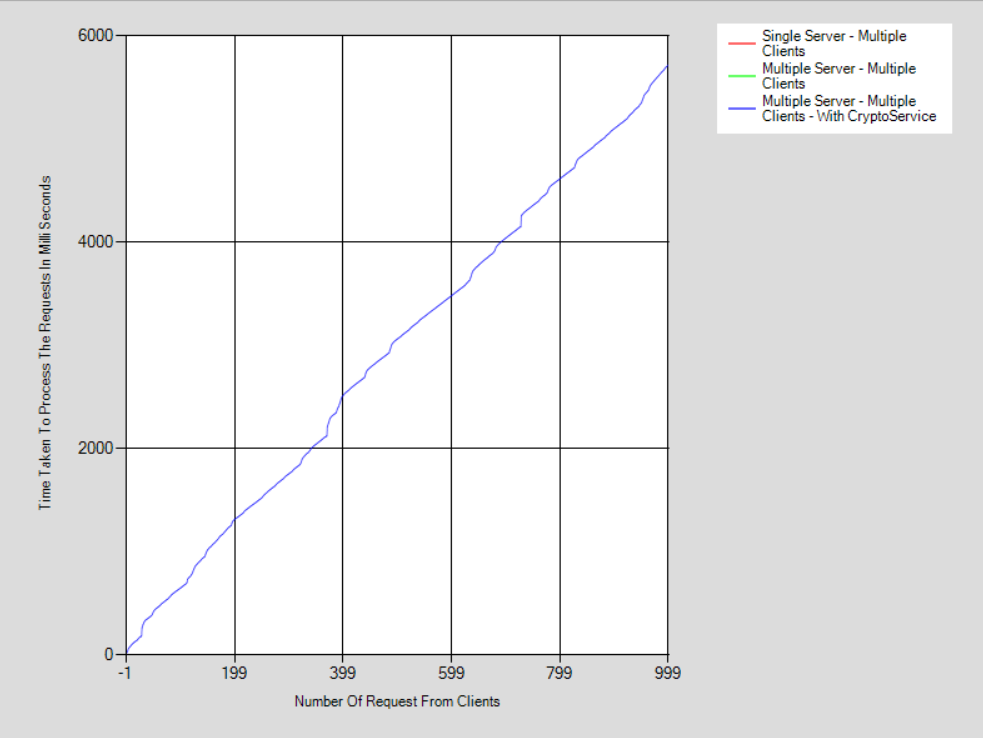
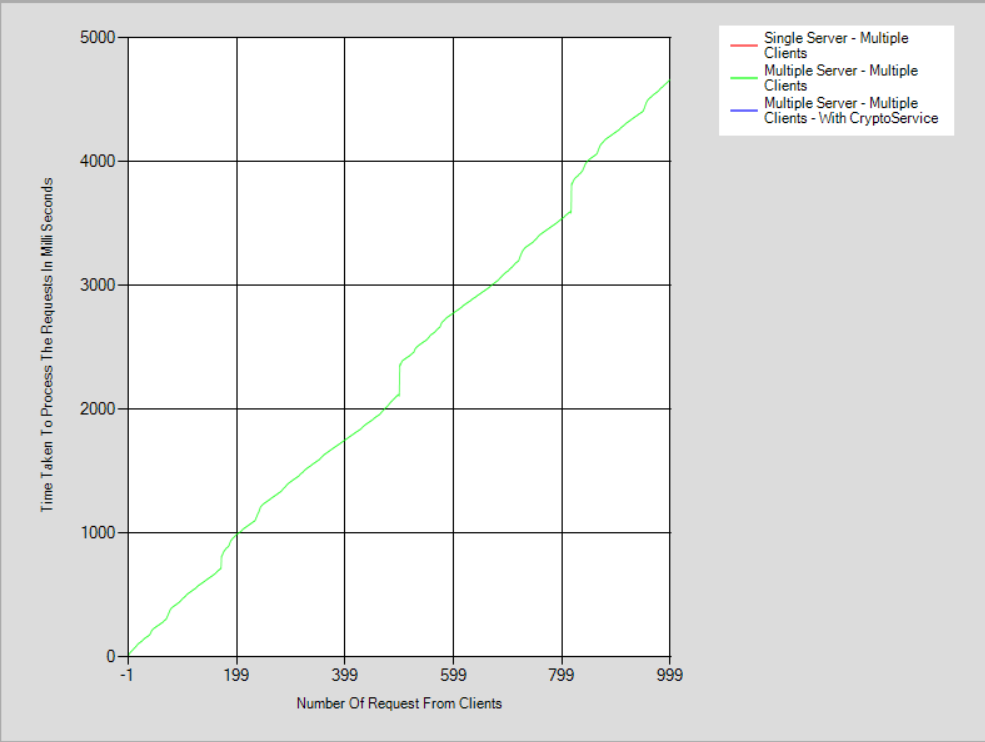
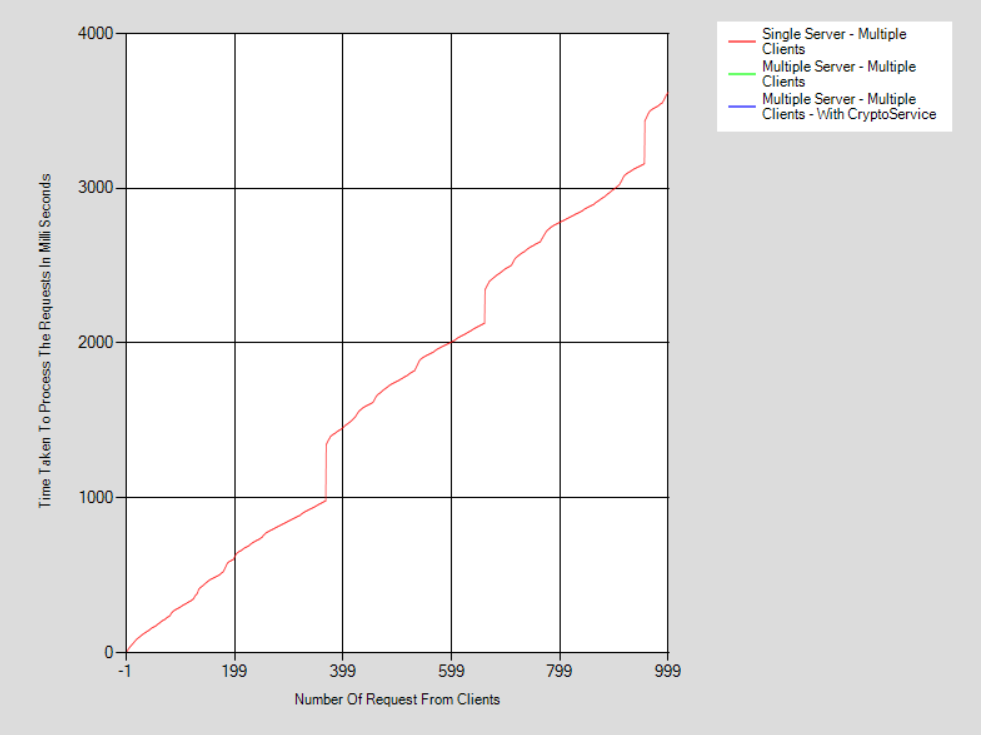
The first scenario has a single server serving multiple clients. We implement the cryptographic service in this scenario to show that the efficiency of using the cryptographic service in the Byzantine Fault Tolerant system. Once the system processes all the requests, it displays the elapsed time on the text box next to that button. In the second scenario, we have multiple scenarios to serve multiple clients. Once the system processes all the requests, it again displays the elapsed time in the text box. Also, in this scenario we don’t have any cryptographic service implemented. The last and final scenario is where we implement the cryptographic service in multiple server – multiple client system. Again, the elapsed time is mentioned on the text box once all the requests are served by the servers. Once we are done with the scenarios, in order to view the database table, we can click on the buttons mentioned in the second layout – Database Info.



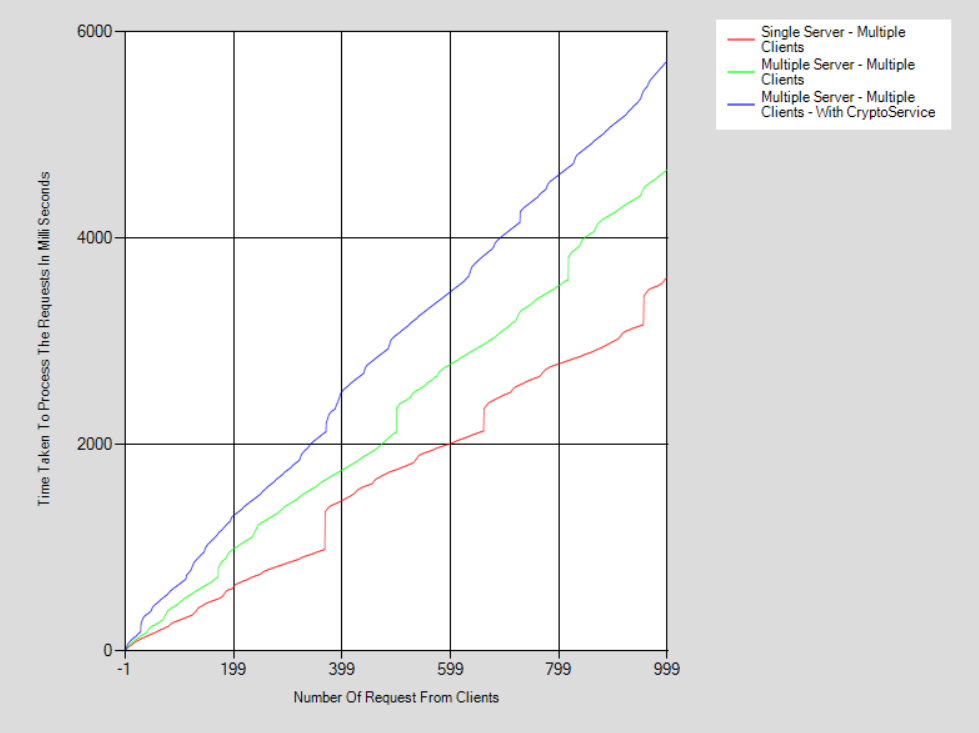
**FIGURE 8. Shows the elapsed time for each scenario.**

**Output**

The output is displayed as a graph for all the scenarios using the button designed for their respective scenario. Finally, the last button is used to compare all the results.



**FIGURE 9. Shows the graph for each scenario.**



**FIGURE 10. Graph comparing all the scenarios.**

The above output consolidates all the graphs which are generated from different scenarios. From this we can see that the red marker shows that the single server with multiple clients takes the least amount of time in processing all the client requests. The other important thing to note is that we have implemented the RSA algorithm in this scenario which shows that the use of RSA doesn’t decrease the performance of the system but increases the security. The green marker in the graph shows the multiple server – multiple client scenario. We can see that the time taken to process all the client requests has increased when compared with the single server – multiple clients scenario. This loss in time has been compromised by the increase in fault tolerance of the system because SAREK Byzantine Fault Tolerant system is more fault tolerant when compared with the existing Byzantine Fault Tolerant systems. The third marker blue which plots the multiple server – multiple clients scenario shows that the time taken to process all the client requests is the highest when compared to other two scenarios. But on the positive side, we can see that the system is more fault tolerant to Byzantine issues and is more secure than the traditional Byzantine Fault Tolerant systems.

**CHAPTER 7**

**FUTURE IMPROVEMENTS**

From the above conclusion we can infer that SAREK Byzantine Fault Tolerant system with Cryptographic service is more secure and more fault tolerant towards Byzantine issues in the computer systems. The extra time the system takes to process the client requests can be compromised with better fault tolerant system and a secure system. But on the other hand, we would find better ways of securing the communication between the client and server by implementing other ideas. Finding a better way helps in reducing the processing time between the second and third scenario which would help in implementing a better system.

**CHAPTER 8**

**CONCLUSION**

In this project we started with understanding of what a Byzantine problem is and then we understood how the existing Byzantine Fault Tolerant system. We then proposed a new system which enhances the performance and security of the existing Byzantine Fault Tolerant system. The implementation of SAREK Byzantine Fault Tolerant system showed us that though the time taken to process the user request increased by a marginal amount, the system is said to be more resistant to Byzantine faults. Moreover, the use of cryptographic service in this SAREK Byzantine Fault Tolerant systems, has improved the security along with the improvisation in performance when compared to existing Byzantine Fault Tolerant systems.

**REFERENCES**

[1] B. Li, W. Xu, M. Z. Abid, T. Distler and R. Kapitza, "SAREK: Optimistic Parallel Ordering in Byzantine Fault Tolerance," 2016 12th European Dependable Computing Conference (EDCC), Gothenburg, 2016, pp. 77-88. doi: 10.1109/EDCC.2016.36. URL: [http://ieeexplore.ieee.org.csulb.idm.oclc.org/stamp/stamp.jsp?tp=& arnumber=7780347&isnumber=7780250](http://ieeexplore.ieee.org.csulb.idm.oclc.org/stamp/stamp.jsp?tp=&%20arnumber=7780347&isnumber=7780250)

[2] Cryptographic Service used while implementing in Visual Studio - https://msdn.microsoft. com/en-us/library/system.security.cryptography.rsacryptoservice provider (v=vs.110).aspx

[3] RSA Algorithm reference 1: <https://en.wikipedia.org/wiki/RSA_(cryptosystem)>

[4] RSA Algorithm reference 2: <https://simple.wikipedia.org/wiki/RSA_algorithm>