Volume 4, Issue 12, December 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Byzantine Fault Tolerant Replication

Bahubali M. Akiwate, Srilaxmi S Dodamani, Nimesh Kumar Paliwal, Ashwini Khandekar, Pratima Bansode
Department of Computer Science and Engineering,
K.L.E College of Engineering and Technology,
Chikodi, Karnataka, India

Abstract- Online tracking using membership service which runs automatically to avoid the human errors which is called as Byzantine fault- tolerant and reconfigurable. Now a day's Byzantine (i.e. arbitrary) faults occur as a result of software errors and malicious attacks; they are increasingly a problem as people come to depend more and more on online services. Systems that provide critical services must behave correctly in the face of Byzantine faults. Correct service in the presence of failures is achieved through replication.

Byzantine-fault-tolerant replication enhances the availability and reliability of Internet services that store critical state and preserve it despite attacks or software errors. However, existing Byzantine-fault-tolerant storage systems either assume a static set of replicas, or have limitations in how they handle reconfigurations (e.g., in terms of the scalability of the solutions or the consistency levels they provide). This can be problematic in long-lived, large-scale systems where system membership is likely to change during the system lifetime. Here there is a complete solution for dynamically changing system membership in a large-scale Byzantine-fault-tolerant system. We present a service that tracks system membership and periodically notifies other system nodes of membership changes. The membership service runs mostly automatically, to avoid human configuration errors; is itself Byzantine fault- tolerant and reconfigurable; and provides applications with a sequence of consistent views of the system membership.

Keywords— Byzantine-fault-tolerant replication, Membership Service(MS), dBQS

I. INTRODUCTION

Today we are more and more dependent on Internet services, which provide important functionality and store critical state. These services are often implemented on collections of machines residing at multiple geographic locations such as a set of corporate data centers. In Existing System, replication enhanced the reliability of internet services to store the data. The preserved data should be secured from software errors. But existing Byzantine-fault tolerant systems is a static set of replicas. It has no limitations. So, scalability is inconsistency. So, these data are not come for long-lived systems.

Our solution has two parts. The first is a MS that tracks and responds to membership changes. The MS works mostly automatically, and requires only minimal human intervention; this way we can reduce manual configuration errors, which are a major cause of disruption in computer systems. The second is storage system dBQS that provides Byzantine-fault-tolerant replicated storage with strong consistency.

II. OBJECTIVES

- Introducing the MS that can reduce manual configuration errors and also a storage system, dBQS that provides Byzantine-fault-tolerant replicated storage with strong consistency.
- Online tracking using membership service which runs automatically to avoid the human errors which is called as Byzantine fault- tolerant and reconfigurable.
- Wireless Mesh Network is a promising technology and is expected to be widespread due to its low investment feature and the wireless broadband services it supports, attractive to both service providers and users.
- Anonymity & privacy issues focus on investigating anonymity in different context.

III. METHODOLOGY

In this approach, there is a complete solution for dynamically changing system membership in a large-scale Byzantine-fault-tolerant system. Byzantine-fault-tolerant replication enhances the availability and reliability of Internet services that store critical state and preserve it despite attacks or software errors. This can be problematic in long-lived, large-scale systems where system membership is likely to change during the system lifetime.

A. Architecture

Figure 1 shows the System Architecture where there are different Local Employees, Remote Employees, & Business Partners and Clients working in the Companies. And Administrator is the one who controls all the functions of the Companies. In this System Administrator provides different services to the clients. Each client should register to access the services like Node Analyser, File download, & File upload. As soon as client get register administrator provides Password and secret key for the clients so that client can use those services. If any client tries to misbehave with the

system then he is considered as a fraud and automatically the prompt will appear on the window that "No privilege to access and your system will get shutdown in less than a minute". In this way the system is reconfigured to avoid human configuration errors nothing but Byzantine faults.

Figure 1: System Architecture

B. Modules

Byzantine fault tolerant replication mainly has two modules with four sub modules which perform specified tasks shown in figure 2. Those are listed below:

- **Admin (Cloud Manager):** It is the main module which provides different kinds of MS to the users such as File upload, File download, & Node analyzer. It also generates password and secret key for different users.
- **User (Remote Client):** In this module user can Login using password and then the user can access MS using secret key provided by Admin.

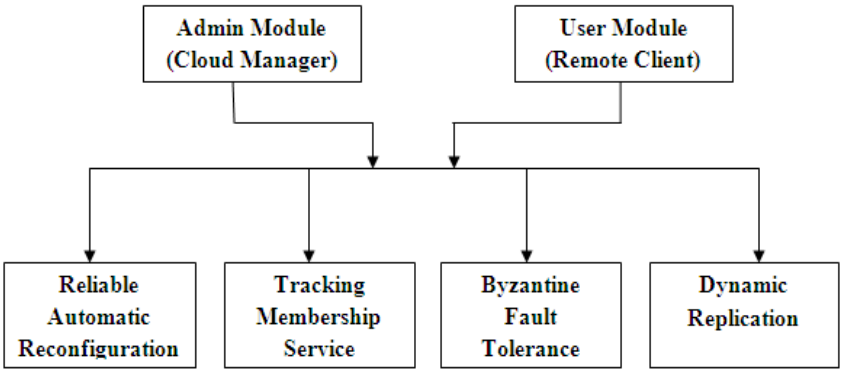

Figure 2: Modules

Reliable Automatic Reconfiguration

In this Module, it provides the abstraction of a globally consistent view of the system membership. This abstraction simplifies the design of applications that use it, since it allows different nodes to agree on which servers are responsible for which subset of the service. It is designed to work at large scale, e.g., tens or hundreds of thousands of servers. Support for large scale is essential since systems today are already large and we can expect them to scale further. It is secure against Byzantine (arbitrary) faults. Handling Byzantine faults is important because it captures the kinds of complex failure modes that have been reported for our target deployments.

Tracking membership Service

In this Module, We assume nodes are connected by an unreliable asynchronous network like the Internet, where messages may be lost, corrupted, delayed, duplicated, or delivered out of order. While we make no synchrony assumptions for the system to meet its safety guarantees, it is necessary to make partial synchrony assumptions for liveness. The MS describes membership changes by producing a configuration, which identifies the set of servers currently in the system, and sending it to all servers. To allow the configuration to be exchanged among nodes without possibility of forgery, the MS authenticates it using a signature that can be verified with a well-known public key.

Byzantine Fault Tolerance

In this Module, to provide Byzantine fault tolerance for the MS, we implement it with group replicas executing the PBFT state machine replication protocol. These MS replicas can run on server nodes, but the size of the MS group is small and independent of the system size. So, to implement from tracking service:

- **Add**: It takes a certificate signed by the trusted authority describing the node adds the node to the set of system members.
- **Remove**: It also takes a certificate signed by the trusted authority that identifies the node to be removed. And removes this node from the current set of members.
- **Freshness**: It receives a freshness challenge; the reply contains the nonce and current epoch number signed by the MS.
- **Probe**: The MS sends probes to servers periodically. It serves respond with a simple ack, or, when a nonce is sent, by repeating the nonce and signing the response.
- **New Epoch**: It informs nodes of a new epoch. Here certificate vouching for the configuration and changes represents the delta in the membership.

Dynamic Replication

In this Module, to prevent attacker from predicting:

- Choose the random number
- Sign the configuration using the old shares
- Carry out a re-sharing of the MS keys with the new MS members
- Discard the old shares

C. Observations

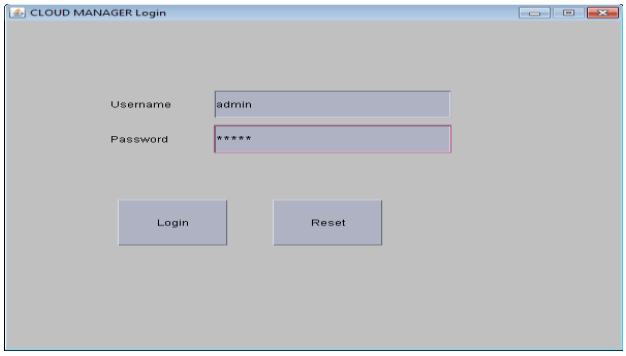

Figure 3: Cloud Manager Login

The above figure 3 represents the Cloud Manager Login, where manager login by adding username and password.

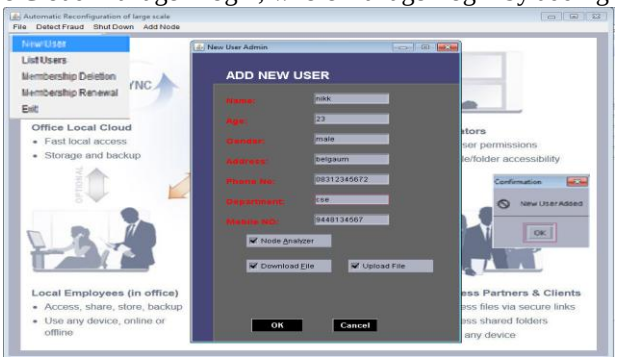

Figure 4: Adding the New User

The above figure 4 shows the Adding New User where user get registered by filling Name, Age, Gender, Address, Phone No., Department, Mobile No and selecting the appropriate services such as Node Analyser, Download File, & Upload File.

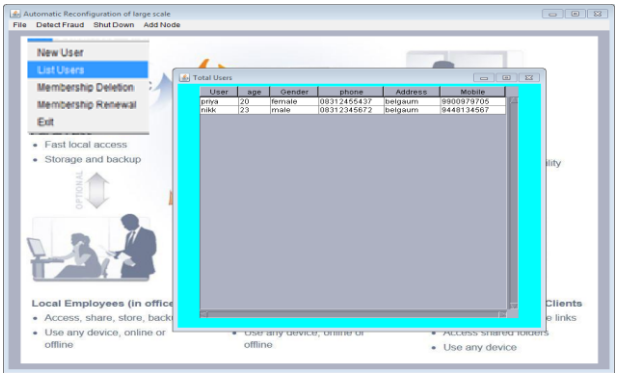

Figure 5: List of Users

The above figure 5 shows the List of users where it represents the total users registered with all their information stored in the database. Even the password and security key is generated by the Cloud Manger it is also stored in database.

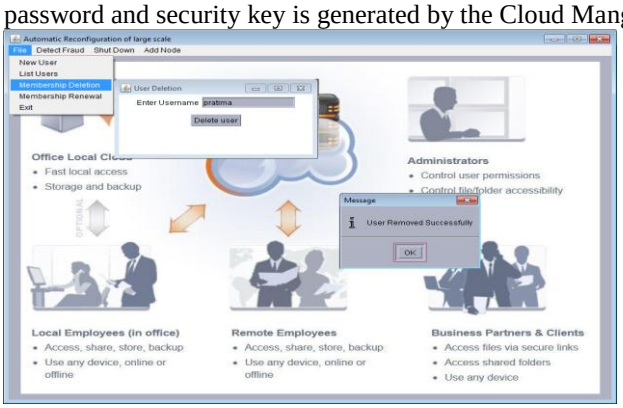

Figure 6: Membership Deletion

The above figure 6 shows Membership Deletion where we enter the username which we want to delete from database by using delete user button. When we click on button the prompt will appear as User Removed Successfully.

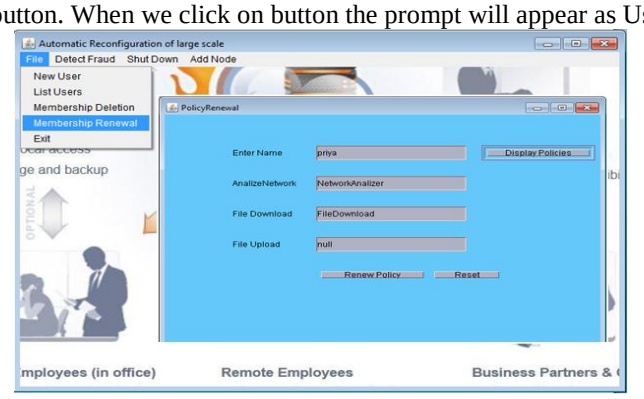

Figure 7: Membership Renewal

The above figure 7 shows the Membership Renewal in which if any user wants to renew or delete his policy then he has to enter his name and can do the appropriate changes under the control of cloud manager.

IV. CONCLUSION

From a proper analysis of positive points and constraints this provides a complete solution for building large scale, long-lived systems that must preserve critical state inspite of malicious attacks and Byzantine failures. We present a storage service with these characteristics called dBQS, and a membership service that is part of the overall system design, but can be reused by any Byzantine-fault tolerant large-scale system.

The membership service tracks the current system membership in a way that is mostly automatic, to avoid human configuration errors. It is resilient to arbitrary faults of the nodes that implement it, and is reconfigurable, allowing us to change the set of nodes that implement the MS when old nodes fail, or periodically to avoid a targeted attack. When membership changes happen, the replicated service has work to do: responsibility must shift to the new replica group, and state transfer must take place from old replicas to new ones, yet the system must still provide the same semantics as in a static system.

ACKNOWLEDGMENT

In this Internet era, there are tens or hundreds of thousands of servers which are designed at large scale. Support for large scale is essential since systems today are already large and we can expect them to scale further. The number and popularity of large scale internet service such as Google, MSN, and Yahoo have grown significantly in recent years. Such services are poised to increase further in importance as they become the repository for data in ubiquitous computing system. Generally network security is to prevent and monitor unauthorized access misuse and modification of data.

REFERENCES

[1] Rodrigo Rodrigues, Barbara Liskov, Member, IEEE, Kathryn Chen, Moses Liskov, and David Schultz, 'Automatic Reconfiguration for Large Scale Reliable Storage Systems' IEEE Transactions on Dependable and Secure Computing , Vol. 9, No. 2, March/April 2012.

[2] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W.Vogels, "Dynamo: Amazon's Highly Available Key-Value Store," Proc. 21st ACM Symp. Operating Systems Principles, pp. 205-220, 2007.

[3] J. Dean, "Designs, Lessons and Advice from Building Large Distributed Systems," Proc. Third ACM SIGOPS Int'l Workshop Large Scale Distributed Systems and Middleware (LADIS '09), Keynotetalk, 2009.

[4] D. Oppenheimer, A. Ganapathi, and D.A. Patterson, "Why Do Internet Services Fail, and What Can Be Done About It?" Proc. Fourth USENIX Symp. Internet Technologies and Systems (USITS '03), Mar. 2003.

[5] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance," Proc. Third Symp. Operating Systems Design and Implementation (OSDI '99), Feb. 1999.

[6] L. Zhou, F.B. Schneider, and R. van Renesse, "Coca: A Secure Distributed On-Line Certification Authority,"ACM Trans. Computer Systems, vol. 20, no. 4, pp. 329-368, Nov. 2002.

[7] M. Bellare and S. Miner, "A Forward-Secure Digital Signature Scheme," Proc. 19th Ann. Int'l Cryptology Conf. Advances in Cryptology (CRYPTO '99), pp. 431-448, 1999.

[8] R. Canetti, S. Halevi, and J. Katz, "A Forward-Secure Public-Key Encryption Scheme," Proc. Conf. Advances in Crptology (EUROCRYPT '03), pp. 255-271, and 2003.

[9] F. Dabek, M.F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-Area Cooperative Storage with CFS," Proc. 18th ACM Symp. Operating Systems Principles (SOSP '01), Oct. 2001.

[10] Amazon S3 Availability Event http://status.aws.amazon.com/ s3-20080720.html, July 2008.

[11] K. Birman and T. Joseph, "Exploiting Virtual Synchrony in Distributed Systems," Proc. 11th ACM Symp. Operating Systems Principles, pp. 123-138, Nov. 1987.

[12] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," Proc. ACM SIGCOMM, 2001.

[13] M. Reiter, "A Secure Group Membership Protocol," IEEE Trans. Software Eng., vol. 22, no. 1, pp. 31-42, Jan. 1996.

[14] H.D. Johansen, A. Allavena, and R. van Renesse, "Fireflies: Scalable Support for Intrusion-Tolerant Network Overlays," Proc. European Conf. Computer Systems (EuroSys'06), pp. 3-13, 2006.

[15] J. Cowling, D.R.K. Ports, B. Liskov, R.A. Popa, and A. Gaikwad, "Census: Location-Aware Membership Management for Large-Scale Distributed Systems," Proc. Ann. Technical Conf. (USENIX'09), June 2009.

[16] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung, "Proactive Public Key and Signature Systems," Proc. Fourth ACM Conf. Computer and Comm. Security (CCCS '97), pp. 100-110, 1997.

