International Journal For Technological Research In Engineering
Volume 3, Issue 8, April-2016 ISSN (Online): 2347 - 4718

IMPLEMENTING RECONFIGURATION OF LARGE SCALE RELIABLE STORAGE SYSTEM BY USING DBQS

E.Tamizhselvi[1], M.Sanjheeviraaman[2]
[1]M.C.A Student, [2]Assistant Professor,
Dept of Computer Applications, Shanmuga Industries Arts and Science College,
Tiruvannamalai, Tamil Nadu, India

ABSTRACT: At present we live in a world of internet services such as email, social networks, web searching, and more which must store increasingly larger volumes of data. These services must run on cheap infrastructure, hence they must use distributed storage systems, and they have to provide reliability of data for long periods as well as availability. Byzantine-fault-tolerant replication mainly used in Internet services that store critical state and preserves it despite attacks or software errors, which enhance the availability and reliability of the services. In existing system we 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).

Reconfiguration handling is one of the limitations; there is a huge problem in long-lived large scale systems, whenever there is a variation in the membership in the entire lifetime of the system. In proposed system, we present a solution for dynamically change in membership for a large-scale Byzantine-fault-tolerant system. A service tracks membership and notifies other system nodes periodically, which runs mostly automatically to avoid human configurations errors. We implement this membership service using distributed hash table called DBQS that provide semantics even changes in replica sets. The membership service is able to manage a large system and the cost to change the system membership is and consumption of the power is low.
Keywords :Byzantine-fault-tolerant, Cloud computing, Data authentication, privacy control, Service of the membership, Membership of the dynamic system.

I. INTRODUCTION

The number and popularity of largescale 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.

Here we use membership service under network security concept to maintain and carried out the storage system service. Now a day's Byzantine fault tolerant system assumes only static set of replicas and it is having limitation in handling the reconfiguration.

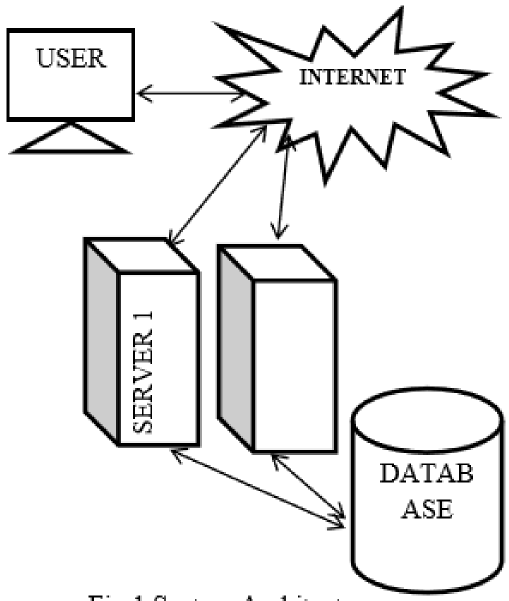


Fig 1 System Architecture

The system is classified into two parts as membership service (MS) and DBQS. One is to monitor the membership changes and the other is to automatically reconfigure the system. Automatic reconfiguration is mainly done in MS to avoid human configuration errors As an Byzantine-Fault-tolerant[1] group is designed for this reconfiguration and our results shows that the MS is able to manage the storage system of replicas with low cost, we present a storage system DBQS that provides Byzantine-fault-tolerant replicated storage with strong consistency. Practical Byzantine fault tolerance (PBFT)[1],describes a new replication algorithm that tolerates Byzantine fault (asynchronous environment better performance) the algorithm provides safety if it does this by sending eviction messages to other MS replicas and then waiting for signed statements from at least MS replicas(including itself). Other MS replicas accept and sign a statement saying so if their last n evicting for that node have failed where n evict < n propose because the initiation of the eviction waited a bit longer than necessary. Most eviction proposals will success if the node is really down, all non-faulty replicas agree on the sequence numbers of requests that commit locally. It provides replicas must change view if they are unable to execute request replicas probe independently and a replica proposes an eviction for a server node that has missed propose probe responses.

II. PRACTICAL BYZANTINE FAULT TOLERANCE

A new replication algorithm that is able to tolerant Byzantine faults. We implemented a Byzantine fault- tolerant NFS service using our algorithm and measured its performance. The results show that our service is only 3% lower than standard replicated NFS. This paper presents a new practical algorithm for state machine replication that tolerates Byzantine faults. This means that clients eventually receive replies to their requests and those replies are correct according to linearizability.

We use cryptographic techniques [2] to prevent spoofing and replays and to detect corrupted messages. Our messages contain public-key signatures, message authentication codes [3] and message digest produced by collision-resistant hash functions. Our algorithm is not vulnerable to this type of attack because it does not rely on synchrony for safety. It improves the performance of Rampart and secure by more than an order of magnitude.

Asynchronous verifiable secret sharing and proactive: Verifiable secret sharing is an important primitive in distributed cryptography [4]. This paper proposes the first practical verifiable secret sharing protocol for asynchronous networks. The protocol creates a discrete logarithm-based sharing and uses only a quadratic number of messages in the number of participating servers.

The second part of this paper introduces proactive cryptosystems in asynchronous networks and represents an efficient protocol for refreshing the shares of a secret key for discrete algorithm [5] based sharing. Finally we propose an efficient proactive refresh protocol and on a randomized asynchronous multi-valued Byzantine agreement primitive.

III. IMPLEMENTATION

3.1 Reliable Automatic Reconfiguration:

In this module, it provides the abstracts of a globally consistent view of the system membership. This abstraction simplifies the design of application that uses it, since it allows different nodes to agree on which server is 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 system today is 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 nodes that have been reported for our target requirement.

3.2 Tracking Membership Service:

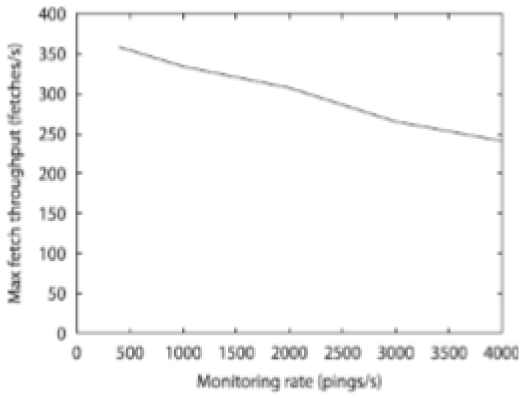
In this module, is only part of what is needed for automatic reconfiguration, we assume nodes are connected by an unreliable asynchronous network like the internet, where messages may be lost, corrupted, delayed, duplicated, out of order. While we make no synchrony assumptions for the system to meet its safety guarantees, it is necessary to make partial synchrony assumptions for liveness. The MS produces configurations periodically rather than after every membership change. The system moves in a succession of time intervals called epochs, and we batch all configuration changes at the end of the epoch. Producing configuration periodically is a key design decision. It allows applications that use the MS to be optimized for long periods of stability. It also permits delayed response to failures, which is important for several reasons: to avoid unnecessary data movement due to temporary disconnections, to offer additional protection against denial of service attacks. Membership Changes: 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 exchange among nodes without possibility of forgery, the MS authenticates it using a signature that can be verified with a well- known public key. The MS assigns each server a unique node ID uniformly distributed in a large, circular ID space which enables the use of consistent hashing to assign responsibility for work in some of our MS protocols. To prevent an attacker from adding a group of servers that are all nearby in the ID space, we require that the node's public key must be chosen by the trusted authority. Probing: The MS detects unreachable servers and marks them as inactive. The MS probes servers periodically using unauthenticated ping messages, which we expect to be sufficient to detect most unreachable servers. However, once a server fails to reply to a signed ping subsequent pings to that server request signatures until a correctly signed response arrives. Ending epochs: To determine when the epoch ends, the MS tracks the termination condition. When the termination threshold is reached, the MS stops probing, and produces an epoch certificate signed by the MS's private key. The signature in the digest of the membership servers and their reach ability status and the epoch number of the new epoch, then the MS sends a NEWEPOCH message to the other servers. This message contains the certificate and new epoch number and describes the configuration changes. It contains the list of added, removed, inactive and reconnected servers. The message is authenticated by the MS so that verifying it is easy. Transferring details is necessary for scalability. Freshness: We provide the freshness by using certificates. Clients of the applications using the MS need to verify the freshness of their configuration to ensure they are communicating with the group that currently stores an item of interest and not an old group. Freshness certificates do not constraint the MS. It moves to the next epoch when thresholds are reached without regard freshness certificates. They also do not prevent clients from moving to a new epoch and a client need no refresh certificate.

IV. PERFORMANCE EVALUATION

This section presents our experimental evaluation various common measures are applied for performance evaluation. This evaluation defines system size, actual number of servers and the time at which a server becomes computed for each server.

4.1 Epoch Performance:

A final point is that the MS can adjust the number of committees dynamically, based system size and proberate.The below figure shows that fetch throughput decreases from 350 fetches/seconds to 250 fetches/seconds as we increase ping load to near maximal.



V. CONCLUSION

Storage system DBQS was implemented and membership services are provided. Membership service is part of the overall project; here the faulty servers can be reused by the BFT system. The membership service works mostly automatic to avoid human configuration errors. When membership changes the replicated service has responsibility to the new replica group and state transfer must take place from old replicas to new. This is accomplished in DBQS. We implemented the membership service and DBQS .Our experiments show that our approach is practical and could be used in a real deployment, the MS can manage a very large number of servers, and reconfigurations have little impact on the performance of the replicated service. In future research, the more committees are needed for the data will be needed in the system size increases. The Membership service can accept the committee dynamically is based on system size and to add extension of our system. The design of a mechanism is determine which machines to place file replicas on the other file to use membership service.

REFERENCES

[1] A. Clement, M. Marchetti, E. Wong, L. Alvisi, and M. Dahlin, "Making Byzantine Fault Tolerant Systems Tolerate Byzantine Faults," Proc.Sixth USENIX Symp. Networked Systems Design and Implementation.

[2] M. Bellare and S. Miner, "A Forward-Secure Digital Signature Scheme," Proc. 19th Ann. Int'l Cryptology Conf. Advances in Cryptology.

[3] R. Canetti, S. Halevi, and J. Katz, "A ForwardSecure Public-Key Encryption Scheme," Proc. Conf. Advances in Cryptology.

[4] J. Cowling, D.R.K. Ports, B. Liskov, R.A. Popa, and A. Gaikwad, "Census: Location-Aware Membership Management for Large-Scale Distributed Systems" [5]M. Reiter, "A Secure Group Membership Protocol,"

Authors:

Ms.E.TamizhSelvi, student, studying in MCA, Department of Computer Applications, Shanmuga Industries Arts and Science College, Tiruvannamalai, Tamil Nadu , India. My research are involves Network, Cloud Computing and Network Security. I did this Journal Paper under the guidance of my project guide Mr. M. Sanjheeviraaman, his motivation is to be too good and i proud to do this paper under his excellence.

Mr. M. Sanjheeviraaman, M.Sc., M.C.A., M. Tech., M.phil., I completed his Bachelor degree(Mathematics) in Arignar Anna Govt. Arts College from University of Madras, M.Sc Mathematics in Distance Education from University of Madras, Master of Computer Applications in Mailam Engineering College from Anna University and also Master of Technology respectively from Bharathidasan University, India. I had one year Industrial Experience, presently I am working as an Assistant professor in Department of Computer Applications at Shanmuga Industries Arts and Science College, Tiruvannamalai, TamilNadu, India. My research interests include Cloud Computing and Cloud based Security.