ISSN: 2277 – 9043
International Journal of Advanced Research in Computer Science and Electronics Engineering (IJARCSEE)
Volume 3, Issue 10, October 2014

Dynamic membership In byzantine fault tolerant system

Miss.Rupali Mahadev Awate, Prof.Rashmi Deshpande

***Abstract*— We present an application that provides a solution for tolerating byzantine faults in distributed system. Reliability of data at very large scale is one of the highest challenges we have to face in distributed systems. Also for availability the data is stored on multiple machines at different locations. To increase the storing capacity we need to scale the system and replace the failed node with new nodes. The existing byzantine fault tolerance systems have a limitation on scalability as well as consistency. In this paper we provide a solution for dynamic system membership change. The system provides membership service, and a change in membership is updated at every site. To avoid the human configuration errors, the membership service is automatic. We stored all the semantics using a hash table dbs to maintain the atomicity. dbs extends the existing byzantine quorum protocol to manage the replication .the theoretical analysis shows that the system is dynamically reconfigurable and byzantine faults tolerant system.**

***Index Terms*—** Byzantine fault tolerance, dynamic system membership, membership service, replication..

I. INTRODUCTION

As we all know today's world is the world of internet, which provides us different functionalities. The services provided by internet applications are implemented by using a bundle of machines connected together positioned at different geographic stations. This system is a large scale distributed system that scales consistently .the system should adapt the membership change and provide efficient and reliable service even though some of the nodes fail. It is necessary to remove the crashed nodes and replace it with new nodes; also we need to add the nodes to the system as the storage requirement increases. So the system needs to be updated regularly. The system work with the membership service which keep updates of changes of nodes in the system. There are many systems available for the membership service and dynamically reconfiguration of nodes.in this paper we have proposed a solution for dynamically changing system membership in byzantine fault tolerant system.

Our approach is different because of following three features:

It provides a reliable and consistent view of membership service. Different nodes in the system collectively agree on the service provided by a particular server.

Since, an Internet is very large system; it is supposed to work on large scale.

As the system designed is automated, it is secure against the arbitrary faults.ie. Faults occurred by human interference .To achieve the unique approach, our proposal for tracking system membership with all three properties has three parts. First is the membership service which includes automatic reconfiguration of nodes and tracking system membership changes. The automatic configuration helps to minimize human intervention and reduces the arbitrary errors.MS publishes a new system membership after a time interval to get the consistent set of servers that are working without fail.[1] This set of server is globally consistent as the client and server nodes decide which service will be provided by which particular server.

We could take advantage of the techniques such as digital signature, public key encryption, and a practical algorithm for byzantine fault tolerance to avoid malicious attacks[2].

The system moves to a new group of servers after reconfiguration. To get the latest change in the system periodical membership tracking is done. The system should respond to the membership change appropriately.

Therefore, the next part of our solution deals with automatic dynamic replication. We used a distributed hash table, as a storage system, dbs, which provides replicated storage of byzantine fault tolerance. Byzantine quorum protocol was originally designed to work with a static set of replicas. We extend it to provide atomic set during replica changes dynamically.

By our implementation, we can say that MS is able to respond to dynamic changes in system membership in large systems.

II. RELATED WORK

There are different systems that keeps track of dynamic system membership, but do not provide all three features provided by our system.

Dynamo: Amazon's Highly Available Key-Value Store[4].Reliability at massive scale is one of the biggest challenges we face at Amazon.com, one of the largest e-commerce operations in the world; even the slightest outage has significant financial consequences and impacts customer trust. The Amazon.com platform, which provides services for many web sites worldwide, is implemented on top of an infrastructure of tens of thousands of servers and network components located in many datacenters around the world. At

Manuscript received Oct , 2014.

***Rupali Mahadev Awate**, M.E.I.TSiddhant College of Engineering,Pune University. sudumbare,pune,India, 7798238766*

***Rashmi Deshpande**, M.E.I.TSiddhant College of Engineering,Pune University, sudumbare,pune,India,*

this scale, small and large components fail continuously and the way persistent state is managed in the face of these failures drives the reliability and scalability of the software systems.

This paper presents the design and implementation of Dynamo, a highly available key-value storage system that some of Amazon's core services use to provide an "always-on" experience. To achieve this level of availability, Dynamo sacrifices consistency under certain failure scenarios. It makes extensive use of object versioning and application-assisted conflict resolution in a manner that provides a novel interface for developers to use.

Chord[8] A fundamental problem that confronts peer-to-peer applications is the efficient location of the node that stores a desired data item. This paper presents Chord, a distributed lookup protocol that addresses this problem. Chord provides support for just one operation: given a key, it maps the key onto a node. Data location can be easily implemented on top of Chord by associating a key with each data item, and storing the key/data pair at the node to which the key maps. Chord adapts efficiently as nodes join and leave the system, and can answer queries even if the system is continuously changing. Results from theoretical analysis and simulations show that Chord is scalable: Communication cost and the state maintained by each node scale logarithmically with the number of Chord nodes.

A Secure Group Membership Protocol[9]

A group membership protocol enables processes in a distributed system to agree on a group of processes that are currently operational. Membership protocol are a core component of many distributed systems and have proved to be fundamental for maintaining availability and consistency in distributed applications. We present a membership protocol for asynchronous distributed systems that tolerates the malicious corruption of group members. Our protocol ensures that correct members control and consistently observe changes to the group membership, provided that in each instance of the group membership, fewer than one-third of the members are corrupted or fail benignly. The protocol has many potential applications in secure systems and, in particular, is a central component of a toolkit for constructing high-integrity distributed services that we are presently implementing

Fireflies[10]: Scalable Support for Intrusion-Tolerant Network Overlays

This paper describes and evaluates Fireflies, a scalable protocol for supporting intrusion-tolerant network overlays. While such a protocol cannot distinguish Byzantine nodes from correct nodes in general, Fireflies provides correct nodes with a reasonably current view of which nodes are live, as well as a pseudo-random mesh for communication. The amount of data sent by correct nodes grows linearly with the aggregate rate of failures and recoveries, even if provoked by Byzantine nodes. The set of correct nodes form a connected submesh; correct nodes cannot be eclipsed by Byzantine nodes. Fireflies is deployed and evaluated on PlanetLab.

Census[3]: Present Census, a platform for building large-scale distributed applications. Census provides a membership service and a multicast mechanism. The membership service provides every node with a consistent view of the system membership, which may be global or partitioned into location-based regions. Census distributes membership updates with low overhead, propagates changes promptly, and is resilient to both crashes and Byzantine failures. We believe that Census is the first system to provide a consistent membership abstraction at very large scale, greatly simplifying the design of applications built atop large deployments such as multi-site data centers.

Census builds on a novel multicast mechanism that is closely integrated with the membership service. It organizes nodes into a reliable overlay composed of multiple distribution trees, using network coordinates to minimize latency. Unlike other multicast systems, it avoids the cost of using distributed algorithms to construct and maintain trees. Instead, each node independently produces the same trees from the consistent membership view. Census uses this multicast mechanism to distribute membership updates, along with application-provided messages.

We evaluate the platform under simulation and on a real-world deployment on PlanetLab. We find that it imposes minimal bandwidth overhead, is able to react quickly to node failures and changes in the system membership, and can scale to substantial size.

III. SYSTEM ASSUMPTIONS

Client-server Model: The system is composed of the number of members that can be servers responsible for storage service and the client node uses that service.

We assumed asynchronous network is connecting the clients and servers. Different kinds of attacks are possible during message transmission; also messages may lose, delayed or duplicated during transmission.

We assume that different cryptographic algorithms, digital signature technique should be used.

We assume that once node is byzantine faulty it remains untrusted forever as its secret information is leaks.

We assume that atomicity, consistency will be provided by the dbs.

IV. MEMBERSHIP SERVICE

A. MS FUNCTIONING

The Membership Service (MS) provides the information about the current members of the system. A configuration message is produced by MS for collecting the information of servers in the system. This information is sent to all servers. The configuration is authenticated by MS by using a signature. The receiving node can verify it by using a public key.

The configuration message is produced by MS after a fix time interval known as an epoch. Every time interval is assigned a sequential number, epnum. Every node in the epoch receives the same configuration. The epnum helps to get the most recent configuration.

Membership change is done by 'Add' and 'remove' Requests. To avoid the Sybil attack [11] and to restrict the entry of malicious servers, the servers are added by using a certificate. The certificate is signed by the trusted authority. While adding a server, the certificate should contain the network address and port number of server. While certificate to Remove server is signed by a trusted authority that server will be removed from a current set of members. After adding, each server is assigned with a unique node ID by MS. The unique node ID is assigned by using the values in the certificate.

MS sends a ping message to the servers that contain nonce. The receiving node signs the nonce in reply. The signed nonce is verified by the MS. If any server fails to reply; it is ping for the threshold number. After that the server is marked as inactive. The unreachable servers are marked as inactive by MS.

If inactive server contacts to MS, it is verified with a challenge message and the marked as reconnect. If any server remains unreachable for a particular no of epoch, it is removed from the system.

B. BFT OPERATION

To achieve Byzantine fault tolerance the BFT state machine replication protocol is used. ADD and REMOVE operations can be implemented as BFT operations. To find the faulty servers BFT operation helps.

V. REPLICATION

Storage applications or services can be extended to handle reconfigurations using the membership service[1]. We present dbs. a read/write block storage system based on Byzantine quorums.MS gives input to DBS so that it can determine when to reconfigure.

dbs. is distributed hash table (DHT) that works on two types of objects. Public-key objects can be written or changed by multiple clients, whereas content-hash object cannot be changed after creation.

To determine data freshness, the data objects are assigned a version number .Each object is stored with a signature that covers both the data and the version number. Signature verification by a public key is done to check the integrity when a client fetches an object.

VI. IMPLEMENTATION

Every object is replicated at 3f+1 replica, numbered from 0 to 3f. Quorums can be any subset with 2f +1 replica[6]. We use a three-phase protocol to write. A prepare certificate contains <PREPARE-REPLY, ts, h>σr in which every statement is authenticated by replica r and contain the same timestamp ts, and hash h.

A valid write certificate contains <WRITE-REPLY, ts>σr. Here replica r authenticates every component and all statements contain the same timestamp ts.

Notation c.ts is used to denote the timestamp in that certificate, and the notation c.h denote the hash in a prepare certificate c.

Each replica keeps: data, the value of the object. Pcert, a valid prepare certificate for data. Plist, containing the timestamp, hash, and client identifier of proposed writes. writeTS, the timestamp of the latest write known to have completed at 2f + 1 replica

A.WRITE PROTOCOL:

Different clients should choose different timestamps, therefore timestamps are constructed by concatenating a sequence number with a client identifier[5][6]: ts = <ts.val, ts.id>.The client identifiers are unique. Client with identifier c uses the function: succ (ts,c) = <ts.val + 1, c> to increment the timestamp. Timestamps compared by comparing val parts and if these agree, comparing the client ids. Figures 1 and 2 give the pseudo code of our three-phase write, for the client and replicas respectively. In all phases, clients retransmit their requests to account for lost messages until they collect a quorum of valid replies.

In Phase2, The replicas check that the timestamp being proposed is correct, client is doing just one prepare, value being proposed does not differ from a previous request for the same timestamp, and that the client has completed its previous write[6].

B.READ PROTOCOL:

The client sends a <READ, nonce> request to all replicas. A replica replies with its value, prepare certificate, and nonce and authenticated it. The client waits for a valid response and chooses the one with the largest timestamp. Read protocol will end if all the timestamp is same. Otherwise the write-back phase for the largest timestamp will be performed by client.

The steps for read and Write Operations

```
Steps at client c to write value val.
• phase 1.
1. Send (READ-TS, nonce) to all replicas
2. Wait for valid, well-formed, and correctly Authenticated replies of the form (READ-TSREPLY,p, nonce)
3. Select the certificate with largest timestamp, Pmax.
• phase 2
1. Send (PREPARE, Pmax, t, h (val), Wcert>σc, authenticated by c, to all replicas. Here t = succ(Pmax.ts, c),and Wcert is the write certificate of c's last write (as explained later) or null if this is c's first write.
2. Wait for a quorum of valid (well-formed, correctly signed, with matching values for h and t) replies of the form _PREPARE-REPLY, t, h>σr. These replies form a
Prepare certificate Pnew for h and t.
• phase 3.
1. Send <WRITE, Val, Pnew> σc to all replicas.
2. Wait for a quorum of valid (well-formed and correctly signed) replies of the form <WRITE-REPLY, t>σr. These replies form a write certificate, which the client retains for its next write.
```

Fig. 1.: Client side read-write operation.

.

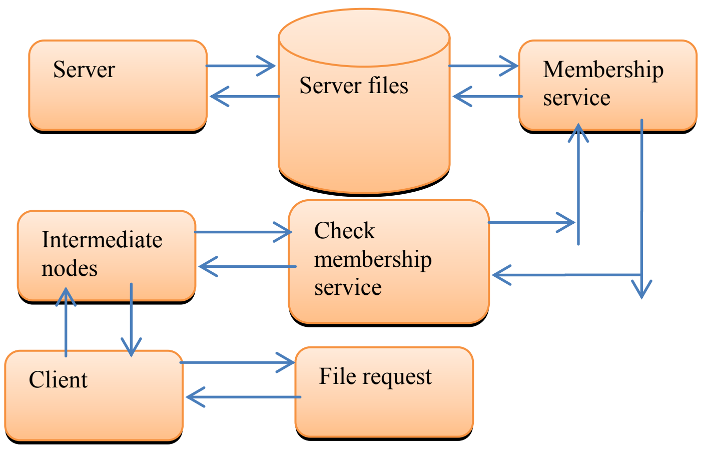

Fig. 3: System Architecture

Steps at replica r to handle the write protocol messages.
• Phase 1.
On receiving <READ-TS, nonce>:
Reply <READ-TS-REPLY, Pcert, nonce>σr
• Phase 2.
On receiving <PREPARE, prepC, t, h, writeC>σc :
1. If request is invalid or t≠succ (prepC.ts, c), discard request without replying to the client.
2. If writeC isn't null, set writeTS =max (writeTS, writeC. Ts), and remove from Plist all entries e such that e.ts ≤ writeTS.
3. If Plist contains an entry for c with a different t or h, discard request without replying to the client.
4. If <c, t, h> isn't already in the Plist, and t > writeTS, add<c, t, h> to Plist
5. Reply <PREPARE-REPLY, t, h>σr.

• Phase 3.
On receiving <WRITE, v, Pnew>σc:
1. If request is invalid (incorrectly authenticated, or signatures In certificate do not verify), or Pnew.h ≠ h(v),discard request without replying to client.
2. If Pnew.ts > Pcert.ts, set data to v and Pcert to Pnew
3. reply <WRITE-REPLY,Pnew.ts>σr .

Fig. 2. Replica read-write operation

VII. CONCLUSION AND FUTURE WORK

This paper presents a complete solution for building dynamic, long-lived distributed systems that must preserve critical state in spite of malicious attacks and Byzantine failures. We present a storage service with these characteristics called dbs, and a membership service that is part of the overall system design, but can be reused by any Byzantine-fault tolerant system.

The membership service tracks the current membership automatically, to avoid human configuration errors. Our system is reconfigurable, which allow us to change the set of nodes that implement the MS when old nodes fail. When membership changes happen, responsibility must shift to the new replica group, and state transfer must take place from old replicas to new ones.

We implemented the membership service and dbs. 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.

FUTURE SCOPE:

Byzantine fault tolerant systems behave correctly when no more than f out of 3f + 1 replica fail[. When there are more than f failures, traditional BFT protocols make no guarantees whatsoever. Malicious replicas can make clients accept arbitrary results, and the system behavior is totally unspecified. We will try to solve this problem[11][12].

ACKNOWLEDGEMENT

Our heartfelt thanks go to Siddhant College of Engineering for providing a strong platform to develop our skills and capabilities. We would like to thank to our guide & respected teachers for their constant support and motivation for us. Last but not least, we would like to thanks all those who directly or indirectly help us in presenting the paper

REFERENCES

[1] Rodrigo Rodrigues, Barbara liskov, member, ieee, Kathryn chen, Moses liskov, and David Schultz. March/april2012 ,"Automatic reconfiguration for large-scale reliable storage systems." ieee transactions on dependable and secure computing, vol. 9, no. 2
[2] j. Cowling, d.r.k. ports, b. Liskov, r.a. popa, and a. Gaikwad, june 2009 "census: location-aware membership management for large-scale distributed systems," proc. Ann. Technical conf. (usenix'09),
[3] C. Y. Lin, M. Wu, J. A. Bloom, I. J. Cox, and M. Miller, "Rotation, scale, and translation resilient public watermarking for images," *IEEE Trans. Image Process.*, vol. 10, no. 5, pp. 767-782, May 2001.
[4] g. Decandia, d. Hastorun, m. Jampani, g. Kakulapati, a.lakshman, a. Pilchin, s. Siva Subramanian, p. Vosshall, and w.vogels, ,2007 "dynamo: amazon's highly available key-value store,"proc. 21st acm symp. Operating systems principles, pp. 205-220.
[5] m. Castro and b. Liskov, february 1999 "practical byzantine fault tolerance," proc. Third symp. Operating systems design and implementation(osdi '99), Feb. 1999.
[6] b. Liskov and r. Rodrigues, 2006 ,"tolerating byzantine faulty clients in a quorum system," proc. 26th ieee int'l conf. Distribute computing systems (icdcs '06).
[7] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H.Balakrishnan, "Chord: A Scalable Applications," Proc. ACM SIGCOMM, 2001.
[8] M. Reiter, "A Secure Group Membership Protocol," IEEE Trans.Software Eng., vol. 22, no. 1, pp. 31-42, Jan. 1996.
[9] 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.
[10] J. Douceur, "The Sybil Attack," Proc. First Int'l Workshop Peer-to-Peer Systems (IPTPS '02), 2002.
[11] J. Li and D. Mazieres, \Beyond one-third faulty replicas in byzantine fault tolerant systems," in Proc. NSDI, 2007.
[12] A study of Byzantine fault-tolerant algorithms Vasileios Papadopoulos University of Ioannina { Computer Science department}