Byzantine Fault Tolerance in Large Scale Reliable Storage System

Dhiraj M. Bochare[1], A. S. Alvi[2]

[1]M.E (I.T.) Scholar Department of Information Technology,
Prof. Ram Meghe Institute of Technology and Research Badnera, Amravati, Maharashtra, India

[2]HOD, Department of Information Technology,
Prof. Ram Meghe Institute of Technology and Research Badnera, Amravati, Maharashtra, India

Abstract: ***Byzantine-fault-tolerant replication enhances the availability and reliability of Internet services that store critical state and preserve it despite attacks and software errors. Existing Byzantine-fault-tolerant storage systems assume a static set of replicas, and have restrictions in how they handle reconfigurations. This paper describes a new replication algorithm with PBFT operation that are able to tolerate Byzantine faults. Byzantine fault- tolerant algorithms will be important because malicious attacks and software errors are increasingly common and can cause faulty nodes to exhibit illogical behavior. BFT state machine replication protocols are quite fast, they don't tolerate Byzantine faults very well. Byzantine faults occur as a result of software errors and malicious attacks.***

Keywords: Byzantine Fault Tolerance, Distributed system, Membership Service, Replica, Liveness

1. Introduction

Byzantine fault tolerant system is having restrictions in handling the reconfiguration and assumes only static set of replicas. Large-scale distributed systems today must adapt to dynamic membership, it will afford competent and reliable service although churn and failures. These systems typically incorporate or rely on some sort of membership service and Byzantine fault tolerant, which provides the application with information about the nodes in the system. The principle of network security is to avoid and monitor unlawful access, exploitation and amendment of data in networking. This is achieved by a group membership protocol which enables processes in a distributed system. Membership protocols are a core component of many distributed system and have proved to be fundamental for maintaining ease of use and steadiness in distributed applications. The replicated state machine approach is a general method for implementing a fault-tolerant service by replicating servers and coordinating client interactions with server replicas [1], [2]. Distributed system technologies have penetrated several areas related with computer control and embedded systems. Distribution needs to be combined with fault-tolerance and real time in some applications. Malicious attacks and software errors are increasingly common in today's environment. The emergent reliance of industry and government on online information services makes malicious attacks more attractive and makes the consequences of successful attacks more serious [3],[4], [5]. Thus, the paper provides the following contributions:

- It describes the state-machine replication protocol that correctly survives Byzantine faults in distributed system.
- It provides the concept of a globally consistent view of the membership service.
- It describes a number of important optimizations that allow the algorithm to realize well so that it can be used in authentic systems.
- It is designed to work at large scale, e.g., tens or hundreds of thousands of servers.
- It is protected against Byzantine arbitrary faults.

2. Byzantine Fault Tolerance (BFT)

The world rapidly interconnected today. More and more important services like business transactions are deployed in network and available anywhere. These services are easily reached by remote devices through the Internet and mobile networks. These services often must access top secret data to provide service. A traditional Byzantine fault-tolerant (BFT) system runs different implementations of the same service on several replicas and ensures that correct computation is performed by correct replicas to mask incorrect replicas. Software for providing access anywhere services may contain bugs, Hackers may take advantage of these bugs to interrupt service and take confidential data. The complexity of real world systems and application software makes it complicated to produce bug-free implementations with existing techniques. A promising approach is to use redundancy to harden services against bugs. We implement it with a group of $3f_{MS} + 1$ replica executing the PBFT state machine replication protocol to provide Byzantine fault tolerance for the Membership Service (MS). These Membership Service replicas can run on server nodes, but the size of the MS group is small and sovereign of the system size. The MS operations are translated to request invocations on the PBFT group. Practical Byzantine Fault Tolerance (PBFT) describes a new replication algorithm that tolerates Byzantine faults and practical asynchronous environment and provides enhanced performance. The algorithm provides safety if all non-faulty replicas agree on the sequence numbers of requests that entrust locally. It provides replicas must change view if they are unable to execute a request. PBFT is a replicated state machine protocol that tolerates up to f Byzantine failures with $3f+1$ replicas. Table 1 summarizes the messages in PBFT [1], [8], [9].

2.1 PBFT Operation

PBFT provides a way to execute operations correctly even though up to f replicas out of 3f + 1 are faulty. It can implement ADD and REMOVE as PBFT operations, it will take as influence the respective certificate. These operations update the current set of members which is the PBFT service state maintained by the MS. Freshness challenges can also be implemented as PBFT operations. The MS does more than performing operations: MS is responsible for probing servers, decides when they are faulty, decides when to end an epoch, and propagates information about the new configuration to all the servers in network. This MS work must be done in a way that prevents faulty members of the MS from causing a malfunction, while at the same time ensuring progress. We avoid problems due to faulty nodes by requiring that $f_{MS} + 1$ MS replicas vouch for any action that is based on nondeterministic inputs. A replica proposes an eviction for a server node that has missed $n_{propose}$ probe responses and replicas probe autonomously. It does this by sending eviction messages to other MS replicas, then waiting

Table 1: Summary of messages in PBFT

Message	*Meaning*
Request	A client sends the request to the leader
Pre-Prepare	The leader proposes the client request
Prepare	The replicas confirm the request proposed by the leader
Commit	The replicas accept the confirmed proposal
Reply	The replicas sends the reply to the client
New-View	The replicas enter a new view

for signed statements from at least $f_{MS} + 1$ MS replicas including itself that agree to evict that node. Other MS replicas accept the eviction and sign a statement saying so if their last n_{evict} pings 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 succeed if the node is really down. Once the replica has composed the signatures, it invokes the EVICT operation, which runs as a normal PBFT operation. This operation has two parameters: the identifier of the node being evicted and a vector containing $f_{MS} + 1$ signature from MS replicas agreeing to evict the node. The operation will fail if there are not enough signatures and they do not verify. We use a comparable scheme for reconnecting servers and ending epochs. The proposer waits until other nodes are likely to agree, collects $f_{MS} + 1$ signature, and invokes the RECONNECT or MOVEEPOCH operation, respectively. After the MOVEEPOCH operation is executed, all MS replicas agree on the membership in the next epoch server nodes for which REMOVE operations have been executed are removed and those for which ADD operations have been executed are added. The MS replicas can produce a certificate describing the membership changes for the new epoch when EVICT and RECONNECT operations mark server nodes as inactive or active [6], [8].

2.2. Algorithm

PBFT is a Byzantine fault tolerant state machine replication protocol. It uses a primary replica to assign each client request a sequence number in the serial order of operations. The replicas run a three-phase agreement protocol to reach agreement on the ordering of each operation. After these they can implement the operation with ensuring dependable state at all non-faulty replicas. The state machine approach is a general method for achievement of a fault-tolerant service by replicating servers and coordinating client connections with server replicas. State machines are defined by services, servers, and most programming language structures for supporting modularity. A state machine consists of state variables, which encode its state, and commands, which transform its state. Each command is implemented by a deterministic program. Execution of the command is infinitesimal with respect to other commands and modifies the state variables and produces some output. A client of the state machine makes a request to execute a command. The request names a state machine, names the command to be performed, and contains any information needed by the command [10], [11].

Algorithm is a form of state machine replication. The service is modeled as a state machine that is replicated across different nodes in a distributed system. Each state machine replica maintains the service state and apparatus the service operations. The replicas move through a succession of configurations called views. The algorithm works roughly as follows:

1) A client sends a request to invoke a service operation to the primary.
2) The primary multicasts the request to the backups.
3) Replicas execute the request and send a reply to the client.
4) The client waits for 1 reply from different replicas with the same result; this is the result of the operation.

2.2.1 The Client

Byzantine Quorum Protocol is used to handle changes in replica set, and it provides strong semantics. Semantics enable them to be reconfigurable while ongoing to provide atomic semantics across changes in the replica set. It includes protocols for read and writes operations and dispensation of messages during replica changes. Each object is stored at n=3f+1 node and quorums consist of any subset containing 2f+1 node. An explanation of the client-side read and writes protocols for two functions are discussed below.

a) Write (data)
It sends messages to the replicas in the group that stores and wait for valid responses, from all server in network. Then it sends messages to all replicas and waits for convincing reply. The write operation for a public-key object will normally has two phases. In the read phase, a quorum of 2f+1 replica is contacted to obtain a set of version numbers for the object.

b) Read (data)
It sends messages to the replicas in the group that stores and wait for convincing responses, from all server in network. Then it sends messages to all replicas and waits for suitable response. Then return data to the user. The client requests the object from all replicas in the read phase to perform a read operation. Normally, there will be

2f+1 valid reply that provide the same version number then the result is the correct response and the operation completes. If the read occurs at the same time as with a write, the version numbers may not agree to provide same version number and the operation incompletes. As shown in Fig. 1. The user get back the data from server for that write-back operation is used, then there is a write-back phase in which the client picks the response with the highest version number, writes it to all replicas, and waits for a reply from a quorum .

2.2.2 Normal-Case Operation

The state of each replica includes the state of the service, a message log containing messages the replica has acknowledged, and an integer denoting the replica's current view. The three phases are pre-prepared, prepare, and commit as shown in Fig. 2. The pre prepare and prepare phases are used to totally order requests sent in the same view even when the primary. These two phases proposes the ordering of requests is defective. The prepared and committed phases are used to ensure that requests that commit are totally ordered across views. In the pre-prepare phase, the primary assigns a sequence number to the request, multicasts a pre-prepare message with piggybacked to all the backups, and appends the message to its log. [4], [5].

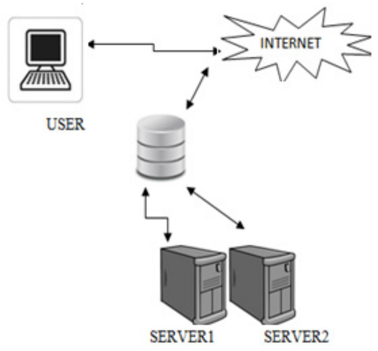

Figure 1: System Architecture

In pre-prepare phase, it assigns sequence number n to request. Multicast pre-prepare message with m piggybacked to all backups and appends the message to its log. The message has the form Message = < < pre-prepare,v,n,d > , m > where v indicates the view in which the message is being sent, m is the client's request message, and d is message m's digest.

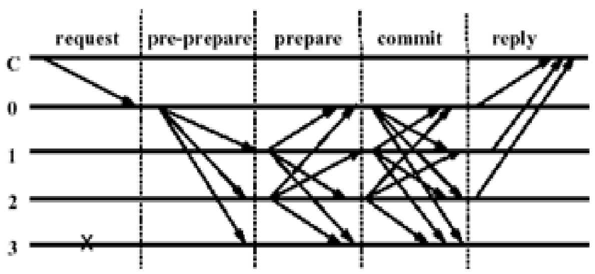

Figure 2: Normal-Case Operation

Replicas accept commit messages and insert them in their log. Signatures provided by replica are same. Let We describe committed and committed-local predicates as follows. Committed (m,v,n) is true, if and only if prepared (m,v,n,i) is true for all i in some set of f+1 non-faulty replicas. Committed-local (m,v,n,i) is true if and only if the replica has accepted 2f+1 commit message from different replicas that match the pre-prepare for m where i indicates replica. Replica i executes the operation requested by m after committed local (m,v,n,i) is true and i's state reflects the sequential execution of all requests with lower sequence numbers. The commit phase ensures the following invariant. If committed-local (m,v,n,i) for some non-faulty i then committed (m,v,n) is true. This ensures that all non-faulty replicas execute request in same order to provide safety property. The algorithm provides safety if all non-flawed commit locally [2], [5], [10].

2.2.3 View Changes

The view-change protocol provides liveness. It allows the system to make improvement when the primary request fails. View changes are triggered by timeouts. View change avoids backups from waiting for an indefinite period for requests to execute. A backup is received a valid request and has not executed it, A backup starts a timer when it receives a request and the timer is not already running. It stops the timer when it is no longer waiting to execute the request. It restarts timer if at that point it is waiting to carry out some other request [3], [4], [5].

3. Optimization

3.1 Reducing Communication

This describes optimizations that improve the performance of the algorithm during normal-case operation. All the optimizations protect the liveness and safety properties. We use three optimizations to reduce the cost of communication.

- The first is to avoid sending most large replies.
- The second reduces the number of message delays for an operation incantation from 5 to 4.
- The third improves the performance of read-only operations that do not modify the service state [5], [10].

3.2 Cryptography

Cryptography is the study of "mathematical" systems involving two kinds of security problems: privacy and authentication. A Privacy system assure the sender of a message that it is being read only by the proposed recipient, it prevents the drawing out information by unlawful parties from messages transmitted over a public channel. An authentication system assures the receiver of a message of the authenticity of its sender, prevents the illegal inoculation of messages into a public channel. Information security uses cryptography to renovate usable information into a form that renders it unusable by anyone other than an authorized user, this process is called encryption. Information that has been encrypted that is rendered unusable can be malformed back into its original usable form by an authorized user, who possesses the cryptographic key, this process is called decryption. The fundamental objective of cryptography is to enable two people to commune over an anxious channel [6], [12]

4. Conclusion

This paper presents a complete solution for building large scale, long lived systems that conserve Byzantine failures by means of PBFT operations. State machine replication is also extensively applicable because it provides strong steadiness, which guarantees that the clients that use the replicated service always observe the same succession of state alter. State machine replication is the most general approach for providing highly accessible stateful services. This paper has described a new state-machine replication algorithm that is able to tolerate Byzantine faults. We have presented competent PBFT algorithms to tolerate crash and Byzantine faults of state machines in distributed systems with PBFT operation. In future Practical Byzantine Fault Tolerance state replication algorithm is practical and could be used in a real deployment, where the MS can manage a very large number of servers.

References

[1] K.Vinothini , Ms. B. Amutha, B.Renganathan, "An Database Query Service In Large Scale Reliable Storage System For Automatic Reconfiguration," IOSR Journal of Computer Engineering (IOSR-JCE) e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 9, Issue 3 (Mar. - Apr. 2013), PP 50 -53

[2] Garg, V.K., "Implementing fault-tolerant services using fused state machines," Technical Report ECE-PDS-2010-001, Parallel and Distributed Systems Laboratory, ECE Dept. University of Texas at Austin (2010)

[3] 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.

[4] Jos´e Rufino, Paulo Ver´ıssimo, Guilherme Arroz, "Node Failure Detection and Membership in CANELy" Proceedings of the 2003 International Conference on Dependable Systems and Networks (DSN’03)

[5] Miguel Castro and Barbara Liskov, "Practical Byzantine Fault Tolerance," Appears in the Proceedings of the Third Symposium on Operating Systems Design and Implementation, New Orleans, USA, February 1999

[6] Dr. B. G. Geetha, M.E., Ph.D, P. Senthil Raja, M.E., (Ph.D), R. Vijay Sai, (M.E.), "Automatic Reconfiguration for Large-Scale trustworthy Storage Systems," Geetha et al. / IJAIR Vol. 2 Issue 4 2013

[7] Miguel Castro, Barbara Liskov, “Practical Byzantine Fault Tolerance and Proactive Recovery" ACM Transactions on Computer Systems, Vol. 20, No. 4, November 2002, Pages 398–461.

[8] 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

[9] Mao, Yanhua, "State machine replication for wide area networks" 2010

[10] Benjamin Wester, James Cowling, Edmund B. Nightingale, Peter M. Chen, Jason Flinn, and Barbara Liskov. "Tolerating latency in replicated state machines through client speculation," In Proceedings of the 6th USENIX symposium on Networked systems design and implementation, pages 245–260, Berkeley, CA, USA, 2009. USENIX Association.

[11] Fred B. Schneider, "Replication Management using the State Machine Approach" ACM Computing Surveys 22 (Dec. 1990).

[12] Whitfield Diffie and Martin E. Hellman, "New Directions in Cryptography" Portions of this work were presented at the IEEE Information Symposium on Information Theory in Ronneby, Sweden, June 21-24 1976

Author Profile

Mr. Dhiraj M. Bochare is Student of second year M.E. (Information Technology) at Prof. Ram Meghe Institute of Technology and Research Badnera, Amravati, Sant Gadge Baba Amravati University, Amravati, Maharashtra, India

Dr. A. S. Alvi is Head of Department, Department of Information Technology at Prof. Ram Meghe Institute of Technology and Research Badnera, Amravati. He has completed his PhD in CSE. - Sant Gadge Baba Amravati University, Amravati, Maharashtra, India