A membership service for dynamically changing large scale storage system

K.VINOTHINI
M.E. (CSE),
SRINIVASAN ENGG COLLEGE,
PERAMBALUR

B. AMUTHA
AP/ CSE,
SRINIVASAN ENGG COLLEGE,
PERAMBALUR

ABSTRACT

Dynamically changing system membership in a large scale reliable storage system is maintained and carried out by a membership service to overcome the current system limitations in handling reconfigurations for a replica set and it is also difficult for life time membership.This service is done with an automatic reconfiguration. Byzantine Fault tolerant replication enhances the availability and reliability of internet service that store critical state and preserve it despite attacks and software errors which provides consistency level. This reconfiguration is carried out by a membership service and dBQS[database Query Service]. dBQS is interesting in its own right because its storage algorithms extend existing Byzantine Quorum protocols to handle changes in the replica set, and it differ from previous DHTs by providing Byzantine Fault tolerance and offering strong semantics. We develop two heuristic algorithms for the problems. Experimental studies show that the heuristic algorithms achieve good performance in reducing communication cost and are close to optimal solutions.

Index terms- Byzantine fault tolerant, PBFT, DHT, Reconfiguration

1. INTRODUCTION

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, lar ge-scale systems where system membership is likely to ch ange during the system lifetime. In this paper, we present a complete solution for dynamically changing system membership in a large-scale.

Byzantine fault tolerant system. It presents 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 itself Byzantine fault-tolerant and reconfigurable and provides applications with a sequence of consistent views of the system membership.

It demonstrate the utility of this membership service by using it in a novel distributed hash table called dBQS that provides atomic semantics even across changes in replica sets. dBQS is interesting in its own right because its storage algorithms extend existing Byzantine quorum protocols to handle changes in the replica set, and because it differs from previous DHTs by providing Byzantine fault tolerance and offering strong semantics.

A **Byzantine fault tolerance** is one which tolerates the byzantine fault. A Byzantine Fault[2] is an incorrect operation (algorithm) that occurs in a distributed system that can be classified as: Omission Failure – a failure of not being present such as failing to respond to a request or not receiving a request. dBQS is a storage system, that provides Byzantine-fault-tolerant replicated storage with strong consistency. dBQS serves as an example application that uses the membership service and takes advantage of its strong consistency guarantees.

Additionally, dBQS is important on its own for two reasons. First, to develop dBQS we had to extend existing Byzantine quorum protocols[2], originally designed for a static replica set, to enable them to be reconfigurable while continuing to provide atomic semantics across changes in the replica set. Second, dBQS implements the popular DHT interface but differs from previous DHTs by handling Byzantine faults and focusing on strong semantics, which can facilitate design of applications that build on a DHT interface.

Distributed Hash Table (DHT) is a distributed[1] and often decentralized mechanism for associating hash values (keys) with some kind of content. Participants in the DHT each store a small section of the contents of the hash table.

The main advantage of DHTs is their scalability. Membership Service 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.

2. RELATED WORK

A group membership protocol enables processes in a distributed system to agree on a group of processes that are currently operational. Membership protocols are a core component of many distributed system and have proved to be fundamental for maintaining availability and consistency in distributed applications. We present am membership protocol for asynchronous distributed system that tolerates the malicious corruption of group members. Our protocols ensure that correct members control and consistency 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 secure and fault-tolerant distributed services that we have implemented. The Membership protocol implementation for an atomic broadcast facilitates a set of techniques that make such a tool kit practical.

Membership protocol[11] is suitable for used in distributed system in which some process may be corrupted by a malicious intruder. This protocol achieves is an asynchronous system, provided that in each instance of the group membership, fewer than one-third of the group members are corrupted or fail. Although the protocol can be used to remove them from the group once detected.

The main drawbacks of the protocols are their relatively large round complexity for group merge operations. This approach is that the resulting protocols are not optimal in their performance, i.e., the compiler adds a certain overhead of additional messages which do not seem necessary. Therefore, there use in real system can be facilitated by providing members with consistent group membership information that cannot be manipulated by corrupt members.

In order to achieve availability in the presence of failures, the objects are replicated. In order to maintain memory consistency in the presence of small and transient changes, the algorithm uses configurations, each of which consists of a set of members plus sets of read-quorums and write-quorums.

In order to accommodate larger and more permanent changes, the algorithm supports reconfiguration, by which the set of members and the sets of quorums are modified. Such changes do not cause violations of atomicity.

Any quorum configuration may be installed at any time we first provide a formal specification for reconfigurable atomic shared memory as a global service.

The service Rambo[9], which stands for "Reconfigurable Atomic Memory" for Basic Objects The rest of the paper, presents our algorithm and its analysis. The algorithm carries out three major activities, all concurrently: reading and writing objects, introducing new configurations, and removing ("garbage-collecting") obsolete configurations.

The algorithm is composed of a main algorithm, which handles reading, writing, and garbage-collection, and a global reconfiguration service, Recon, which provides the main algorithm with a consistent sequence of configurations. Reconfiguration is loosely coupled to the main algorithm, in particular, several configurations may be known to the algorithm at one time, and read and write operations can use them all.

3.RECONFIGURATION TECHNIQUE

3.1 STORAGE

Byzantine Quorum Protocol, It's used to handle changes in replica set, and it provides strong semantics. That is, to enable them to be reconfigurable while continuing to provide atomic semantics across changes in the replica set. It includes protocols for read and writes operations and processing of messages during replica changes. Each object is stored at n=3f+1 nodes and quorums consist of any subset containing 2f+1 nodes.

A description of the client-side read and writes protocols for 2 functions are shown below,

Write (data)

Send messages to the replicas in the group that stores and wait for valid responses, all for server. Then send messages to all replicas and wait for valid response.

The write operation for a public-key object is normally has two phases. In the read phase, a quorum of 2f+1replicas is contacted to obtain a set of version numbers for the object.

Read (data)

Send messages to the replicas in the group that stores and wait for valid responses, all for server, send messages to all replicas and wait for valid response. Then return data to the user.

To perform a read operation, the client requests the object from all replicas in the read phase. Normally, there will be 2f+1 valid reply that provide the same version number; in this case the result is the correct response and the operation completes. However, if the read occurs concurrently with a write, the version numbers may not agree. As shown in fig 1.1 the user get back the data from server for that write-back operation is used. In this case, 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.

3.2 PBFT

Practical Byzantine Fault Tolerance (PBFT), it describes a new replication algorithm that tolerates Byzantine faults and practical (asynchronous environment, better performance). The algorithm provides safety if 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 a request.

Replicas probe independently, and a replica proposes an eviction for a server node that has missed p_{ropose} probe responses. It does this by sending eviction messages to their MS replicas, and then, waiting for

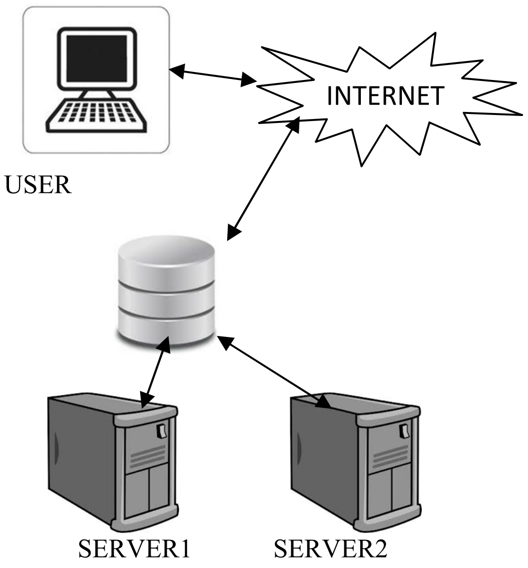

Fig 1.1:System Architecture

signed statements from at least f_{MS}+1 MS replicas (including it) 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 $e_{vict}<n_{propose}$.Because the initiation of the eviction waited a bit longer than necessary, most eviction proposals will succeed if the node is really down.

4. EXPERIMENTAL RESULT

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.

Throughout our experiments, we tried to determine how system components interfere with each other. For this experiment, the replica doing the pings was associated with an instance of dBQS (since we intend to run committees on system nodes).

We repeated the experiment under three different degrees of activity of the dBQS server: when it is not serving any data (which will be the case when the MS does the pings and runs on special nodes that don't also handle the application), when it is handling 30 queries/second, and when clients saturate the server with constant requests,which leads to the maximal number of about 300 queries/second. Each query requested a download of a 512 byte block.little impact on the performance of the replicated service.

5. CONCLUSION AND FUTURE WORK

A dynamically changing system membership in a large scale reliable storage system is maintained and carried by a special service.For that we provide a membership service that track the system periodically and notifiy the changes by using storage and PBFT algorithm.By which the system will reconfigured automatically.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 to determine which machines to place file replicas on the other file to use membership service.

REFERENCES

[1] Birman. K and T. Joseph, "Exploiting Virtual Synchrony in Distributed Systems," Proc. 11th ACM Symp. Operating Systems Principles, pp. 123-138, Nov. 2007.

[2] Clement. A, M. Marchetti, E. Wong, L. Alvisi, and M. Dahlin,"Making Byzantine Fault Tolerant Systems Tolerate ByzantineFaults," Proc. Sixth USENIX Symp. Networked Systems Design and Implementation (NSDI '09), Apr. 2009.

[3] Cowling. J, 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] Chen.K., "Authentication in a Reconfigurable Byzantine Fault Tolerant System," master's thesis, Massachusetts Inst. of Technology, July 2004.

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

[6] Dabek.F., 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.

[7] Douceur.J., "The Sybil Attack," Proc. First Int'l Workshop Peer-to-Peer Systems (IPTPS '02), 2002.

[8] DeCandia.G, 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.

[9] Lynch. N. and A.A. Shvartsman, "Rambo: A Reconfigurable Atomic Memory Service," Proc. 16th Int'l Symp. Distributed Computing (DISC '02), 2002.

[10] Rodrigo Rodrigues, Barbara Liskov, Member, IEEE, Kathryn Chen, Moses Liskov, and David Schultz "Automatic Reconfiguration for Large-Scale Reliable Storage Systems"-IEEE transaction on dependable and secure computing.

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