ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 4, April 2014

To Manage A Scalable Distributed System Using Dynamic Configuration

M.Deepika[1], R.Sumathi[2]

M.Deepika, PG scholar-(ME-CSE), Gnanamani College of Technology, Namakkal, T.N, India[1]

R.Sumathi, AP/CSE, Gnanamani College of Technology, Namakkal, T.N, India[2]

Abstract: This paper presents a solution for distributed system that must preserve critical state in spite of malicious attacks and Byzantine failures. In an existing Byzantine replication protocols satisfy the runtime performance. There have been proposals for dynamic replication protocols that tolerate crash failures and provide an efficient runtime performance. We present a new Byzantine fault-tolerant replication protocol that meets the new correctness criterion and evaluate its performance in fault-free execution and when under attack.

Keywords: Byzantine fault-tolerance, distributed system, storage system, system membership.

I. INTRODUCTION

In an existing Byzantine fault-tolerance protocols satisfy the system runtime performance and also system safety and liveness[1]. In this paper, we point out that in many systems, a small number of Byzantine processors can degrade performance to a level far below what would be achievable with only correct processors. Specifically, the Byzantine processors can cause the system to make progress at an extremely slow rate, even when the network is stable and could support much higher throughput. This paper provides a solution to distributed systems. Our approach is unique because

- It provides the abstraction of a globally consistent view of the system membership.
- It is designed to work at large scale. Support for large scale is essential since systems.
- 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.

Our solution has two parts. The first is a membership service (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.

Therefore, the second part of our solution addresses the problem of how to reconfigure applications automatically as system membership changes; we present a storage system, dBQS, which provides Byzantine-fault-tolerant replicated storage with strong consistency.

II. EXISTING SYSTEM

In Existing System, replication enhanced the reliability of internet services to store the data's. Preserved data to 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's are not came for long-lived systems. The existence of the following cryptographic techniques that an adversary cannot subvert: a collision resistant hash function, a public key cryptography scheme, and forward secure signing key and the existence of a proactive threshold signature protocol[2,3].

III. PROBLEM DESCRIPTION

However, existing Byzantine-fault-tolerant systems either assume a static set of replicas or have limitation. So, scalability is inconsistency. This can be problematic in long-lived, large scale systems where system membership is likely to change during the system lifetime.

IV. PROPOSED SYSTEM

In Proposed System, has two parts. The first is a membership service (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 periodically, the MS publishes a new system membership; in this way it provides a globally consistent view of the set of available servers, The choice of strong consistency makes it easier to implement applications, since it allows clients and servers to make consistent local decision about which servers are currently responsible for which parts of the service.

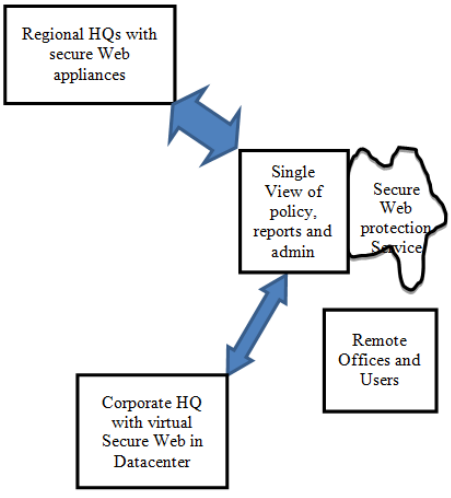

Fig.1.Service Architecture

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 4, April 2014

The second part of our solution addresses the problem of how to reconfigure applications automatically as system membership changes. We present a storage system, dBQS that provides Byzantine-fault-tolerant replicated storage with strong consistency.

V. MS FUNCTIONALITY

A. Membership Change Requests

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 MS responds to requests to add and remove servers. We envision a managed environment with admission control since; otherwise, the system would be vulnerable to a Sybil attack where an adversary floods the system with malicious servers.

Thus, we assume servers are added by a trusted authority that signs certificates used as parameters to these requests. The certificate for an ADD request contains the network address and port number of the new server, as well as its public key, whereas the certificate to REMOVE a node identifies the node whose membership is revoked using its 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; applications can also these IDs if desired.

B. Membership Control

The initial work on group communication systems only tolerated crash failures. Byzantine failures are handled by the Rampart and Secure Ring[4]systems. The membership service has the same goals as the group membership modules present in group communication systems. Adding and removing processes in these systems is a heavyweight operation: all nodes in the system execute a three-phase Byzantine agreement protocol that is introduced by these systems, which scales poorly with system size. We get around this limitation by treating most nodes in the system as clients and using only a small subset of system nodes to carry out the protocol. Thus, our solution is scalable with the number system nodes, which are only clients of the protocols. Guerraoui and Schiper define a generic consensus service in a client-server, crash-failure setting, where servers run a consensus protocol and clients use this service as a building block.

The paper mentions as an example that the servers could be used to track membership for the clients; it also mentions the possibility of the service being implemented by a subset of clients. However, the paper does not provide any details of how the membership service would work. We show how to implement a membership service that tolerates Byzantine faults, and discuss important details such as how to reconfigure the service itself. Peer-to-peer routing overlays can be seen as a loosely-consistent group membership scheme: by looking up a certain identifier, we can determine the system membership in a neighbourhood of the ID space near that identifier. Castro et al. proposed extensions to the Pastry peer-to-peer lookup protocol to make it robust against malicious attacks. Peer-to-peer lookups are more scalable and resilient to chum than our system, but unlike our membership service, do not provide a consistent view of system membership. As a result concurrent lookups may produce different "correct" results. Fireflies is a Byzantine-fault-tolerant, one-hop (full membership) overlay. Fireflies uses similar techniques to ours(such as assigning committees that monitor nodes and sign eviction certificates). However, it does not provide a consistent view of membership, but rather ensures probabilistic agreement, making it more challenging to build applications that provide strong semantic. Census includes a membership service and provides consistent views based on epochs; it builds on the techniques described in these papers for the MS, both to end the epoch and to allow the MS to move in the next epoch. It is designed to work for very large systems, and divides the membership into "regions" based on coordinates. Each region tracks its own membership changes and reports to the MS toward the end of the epoch; the MS then combines these reports to determine the membership during the next epoch and disseminates the changes using multicast.

C. Liveness

To provide maximum period of time or liveness, replicas must move to a new view if they are unable to execute a request. We can achieve these goals by following three ways[5].

First, to avoid starting a view change too soon, a replica that multicasts a view change message for view v+1 waits for 2f+1 and then it will starts the timer after some time T. If the timer expires before it receives a valid new view message for v+1 or before it executes a request in the new view that it had not executed previously, it starts the view change for view v+2 but this time it will wait 2T before starting a view change for view v+3.

Second, if a replica receives a set of f+1 valid view change messages from other replicas for views greater than its current view, it sends a view change message for the smaller view in the set, even if its timer has not expired, this prevents it from starting the next view change too late.

Third, faulty replicas are unable to impede progress by forcing frequent view changes. A faulty replica cannot cause a view change by sending a view-change message ,because a view change will happen only if at least 1 replicas send view-change messages, but it can cause a view change when it is the primary.

D. Safety

The view-change protocol ensures that non-faulty replicas also agree on the sequence number of requests that commit locally in different views at different replicas. A request m commits locally at a non-faulty replica with sequence number in view only if *committed (m,y,n)* is true. This means that there is a set 1 containing at least f+1 non-faulty replicas such that *prepared (m,y,n,i)* is true for every replica in the set. Non-faulty replicas will not accept a pre-prepare for View $v'>v$ without having received a new-view message for (since only at that point do they

enter the view). But any correct new-view messages from every replica in a set R_2 of $2f+1$ replicas. Since there are $3f+1$ replicas, R_1 and R_2 must intersect in at least one replica λ that is not faulty. *k*'s view-change message will ensure that the fact that prepared in a previous view is propagated to subsequent views, unless the new-view message contains a view-change message with a stable checkpoint with a sequence number higher than *n* [6].

E. Correctness

Here, we describe correctness conditions for MS (Membership Service) and dBQS. Those conditions are given below.

(1) *Correctness condition for the MS.* For each epoch e, the MS replica group for e must contain no more than f_{MS} faulty replicas up until the moment when the last non-faulty MS replica finishes that epoch, discards its secret threshold signature share, and advances its forward-secure signing key.

(2) *Correctness condition for dBQS.* For any replica group g_e for epoch e that is produce during the execution of the system, g_e contains no more that *f* faulty replicas up until the later of the following two events: 1) every non-faulty node in epoch e+1 that needs state from g_e has completed state transfer, or 2) the last client freshness certificate for epoch or any earlier epoch expires at any non-faulty client *c* that accesses data stored by g_e [7].

F. Storage System

Ocean Store is a two-tiered system BFT storage system. The primary tier of replicas offers strong consistency for mutable data using the PBFT protocol and the secondary tier serves static data, and thus, has simpler semantics.

There have been proposals for dynamic replication protocols that tolerate crash failures.

VI. METHODOLOGIES

A. Reliable Automatic Reconfiguration

In this Module, it provides the abstraction of 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.

B. 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, 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 will-known public key.

C. 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.

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 acknowledgement 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.

D. Dynamic Replication

In this Module, to prevent attacker from predicting

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

VII. CONCLUSION

This paper presents a complete solution for building large scale, long-lived systems that must preserve critical state in spite of malicious attacks and Byzantine failures. 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.

REFERENCES

[1] Yair Amir and Jonathan Kirsch, "Prime: Byzantine Replication under Attack", July 2011.

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

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

[4] K.Kihlstrom L.Moser,and P.Melliar-Smith, "The Secure Ring Protocols for Securing Group Communication", Proc. Hawaii Int'l Conf. System Sciences,Jan.1998.

[5] M.Castro, "Practical Byzantine Fault Tolerance", PhD dissertation, Massachusetts Inst. of Technology,2001.

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

[7] L.Alvisi, D.Malkhi, E.Pierce, M.Reiter, and R.Wright, "Dynamic Byzantine Quorum Systems", Proc. Int'l Conf. Dependable Systems and Networks (DSN '00), pp. 283-292, June 2000.

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

[9] R.C.Merkle, "A Digital Signature Based on a Conventional Encryption Function", Proc. Conf. Theory and Applications of Cryptographic Techniques on Advances in Cryptology (CRYPTO '87), 1987.

[10] J.P.Martin and L.Alvisi, "A Framework for Dynamic Byzantine Storage", Proc. Int'l Conf. Dependable Systems and Networks (DSN '04), June 2004.