Secure Chain Replication

Berkin Guler, Oznur Ozkasap
Department of Computer Engineering
Koc University, Istanbul, Turkey
{bguler15, oozkasap}@ku.edu.tr

***Abstract*—We propose an approach for enabling secure chain replication that is resilient against crash and arbitrary failures. The main contribution is provided by a service named Elias which acts as a proxy between clients and the chain replication system. Elias is a trusted service that ensures the secure communication between clients and the replication system while detecting any arbitrary failures through cryptographic hashing. Elias also provides reconfiguration for the chain to eliminate the node affected by a crash or arbitrary failure.**

***Index Terms*—Distributed systems; fault tolerance; chain replication; secure replication;**

I. Problem Definition and Motivation

Replication is the most common way of enabling fault tolerance features in a distributed system. Basically, the nodes of the system are repeated in a redundant manner so that if one fails another one could continue the execution. There are two main approaches in replication and these are called as state-machine replication and primary-backup replication. In state-machine replication, every node executes the same request and there is a voting process takes over after all but in primary-backup replication, each client request is executed by the primary replica then the result is sent to backup replicas such that the communication is handled by a single primary replica. However, in Chain Replication (CR) [1] a client request is executed by the head replica and the result is forwarded to other replicas through the body of the chain and in case of any replica affected by an arbitrary failure or exhibiting malicious behavior, the forwarded message becomes susceptible to be altered before reaching to the tail. Consequently, the confidentiality, integrity, and authenticity of the request and the result are exposed since every replica is involved in the communication.

There exist a few studies that address arbitrary failures in a chain communication setup yet they all assume a state machine replication approach. [2] proposes the Aliph protocol for detecting Byzantine failures. [3] proposes the Shuttle protocol and the Olympus service with 2 different Shuttle implementations where the CRC Shuttle requires $f+1$ replicas whereas the HMAC Shuttle requires $2f+1$ replicas to detect Byzantine failures. The most recent study in this context proposes two different protocols named BChain-3 and BChain-5 [4]. Both protocols can tolerate f failures and for this BChain-3 requires $3f+1$ replicas while on the contrary, the BChain-5 requires $5f+1$ replicas.

Despite the chain communication pattern is taken into account in all these studies, they follow the state machine replication as after a client requests for an operation o, while propagating through the chain the o is applied by all replicas and a result r is obtained which conflicts with the original CR protocol definitions that require single execution of the o and the result r obtained in the head node should be applied by all other nodes until reaching to the tail. Since not only the state-machine replication protocols are implemented in real life services but many of these services are also replicated in the primary-backup replication manner, it is necessary to obtain an arbitrary failure-free execution which requires the usage of BSMR-like secure protocols for primary-backup replication. In contrast to prior works, we address the traditional primary backup nature of CR and propose an approach for enabling secure CR supported by a coordinator service named Elias.

II. Elias

The system includes a traditional CR configuration where there are a head, a tail and a body in the replica chain. The proposed **Elias** service extends the traditional CR configuration, enables secure chain replication and tolerates Byzantine failures. In contrast to the traditional setting, the clients communicate with Elias instead of the head of the chain to send its queries. Elias offers both an easy chain reconfiguration when needed and also acts as a trusted node in the system. Elias enforces the public-key encryption system such that clients have to encrypt the queries using Elias' public key before interacting with the system. Elias system overview is demonstrated in Figure 1.

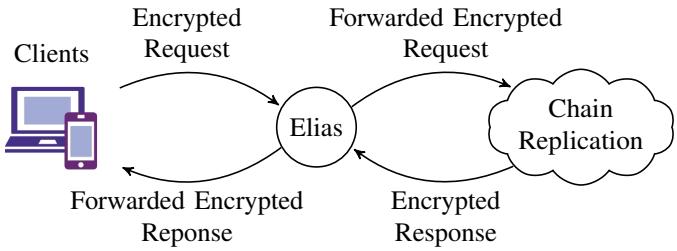

Fig. 1. Elias System Overview

The Elias service algorithm is provided in Algorithm 1. Since Elias acts as a coordinator between the traditional chain replication and the clients, it constantly listens for the incoming client requests. We assume that the replicated service is a distributed key-value store and replicated using the traditional CR protocol. Suppose client c issues an UPDATE operation on the system by the following operation $< K = V >$, which stands for assigning value V to the key K. The actual request will be the following $< \epsilon(K, \beta) = \epsilon(V, \beta) >$ where β denotes

the public key of the Elias and $\epsilon()$ is the asymmetric encryption function.

Once Elias receives the request, it obtains the cryptographic hash value (ψ) of the $\epsilon(V,\beta)$ and saves the $<\epsilon(K,\beta),\psi>$ pair locally (lines 3-4). Then forwards the original request $<\epsilon(K,\beta)=\epsilon(V,\beta)>$ to the head of the chain (line 5). The head updates its $\epsilon(K,\beta)$ field with the $\epsilon(V,\beta)$ value and propagates this change to its successor. Every node of the chain would perform the same operation and apply the result obtained by the head. Once the tail applies the change, it notifies the Elias regarding the change it made and also starts the back propagation of acknowledgments of the traditional CR. When Elias is notified (line 6), it computes the cryptographic hash value of the change and compares it with the pair it has (line 7). If they match, Elias acknowledges the client (line 8) and if they do not, it means that there is an arbitrary failure in a replica.

Elias sends internal READ requests to all replicas of the chain in parallel for the key field $\epsilon(K,\beta)$ by expecting a result whose cryptographic hash is equal to the ψ (lines 10-11). By comparing the hashes it tries to find the closest replica to the head and once finding the failed node, it runs a reconfiguration algorithm on the chain which removes the links from the failed node so the predecessor and the successor of the failed node are now linked to each other (lines 12-15). Eventually, it re-sends the original request of the client to the head to ensure that all replicas are consistent with each other (line 16).

If a client wants to make a READ operation for the key field K it sends the following request to the Elias, $\epsilon(K,\beta)$. Elias looks checks its local database and obtains the cryptographic hash value ψ for the value of the key field $\epsilon(K,\beta)$ (line 18) and then forwards the request to the tail of the chain (line 19). The tail returns the value of the $\epsilon(K,\beta)$ and sends it to the Elias (line 20). The Elias computes the cryptographic hash value of that value and compares it with ψ (line 21). If they are equal, it first decrypts the $\epsilon(V,\beta)$ value with the $\rho(\epsilon(V,\beta),\alpha)$ operation where ρ is the cryptographic decryption function and α is the private key of the Elias. In order to continue the secure communication, Elias again encrypts the result with the following operation $\epsilon(\rho(\epsilon(V,\beta),\alpha),\alpha)$ using its private key so that the client can decrypt it with the public key β of Elias (line 22). However, if Elias detects a mismatch in the cryptographic hash values, it issues a reconfiguration on the chain which eliminates the tail node by letting the predecessor of the tail node that it is the new tail node and Elias forwards the READ request to the new tail node and starts over the aforementioned procedure (lines 26-27). It is possible that Elias had never seen the requested key, so in that case, it cannot check for any arbitrary failure (line 24).

The system with the Elias trusted service promises secure and robust chain replication resilient against arbitrary failures and malicious behavior. However, since Elias stores a private hash database for values that were inserted into the system, if a client wants to read a key that Elias did not witness being inserted into the system, it cannot check for any arbitrary failures and has to trust every replica. Therefore, if the replicated database will contain initial values, a preparation phase could be needed for Elias, to read the keys and values and obtain an initial hash database for them.

Algorithm 1: Elias algorithm

```
1  foreach incoming request from clients do
2      if update request is arrived then
3          ψ = hash(ε(V, β))
4          save pair < ε(K, β), ψ >
5          forward < ε(K, β) = ε(V, β) > to the head
6          wait for ACK from the tail replica
7          if hash(ACK) = ψ then
8              send ACK to client, operation was successful
9          else
10             issue internal read operations to all replicas for
                key, ε(K, β)
11             check if the returned value has the same hash
                value with ψ
12             find the closest replica (ρ_i) to the head, that
                returns the wrong value
13             remove ρ_i out of the chain
14             set the predecessor of ρ_{i+1} as ρ_{i−1}
15             set the successor of ρ_{i−1} as ρ_{i+1}
16             repeat lines 4-7
17     else
18         find pre-saved ψ value for requested key ε(K, β)
19         forward request to the tail replica
20         wait for the value ε(V, β)
21         if ψ exists ∧ hash(ε(V, β))=ψ then
22             send ε(ρ(ε(V, β), α), α) to the client
23         else if ψ does not exist then
24             respond to client, we cannot search for any
                arbitrary failure
25         else
26             remove tail node from the chain, set the
                predecessor of it as the new tail
27             repeat line 21
```

In the poster presentation, we are demonstrating the overall system throughput of both the traditional CR and our proposed method and discuss the performance effects of it by comparing the blocking times seen by clients and the overall throughput over the PlanetLab test platform using the YCSB benchmarking service.

REFERENCES

[1] R. Van Renesse and F. B. Schneider, "Chain replication for supporting high throughput and availability." in *OSDI*, vol. 4, 2004, pp. 91–104.

[2] R. Guerraoui, N. Knežević, V. Quéma, and M. Vukolić, "The next 700 bft protocols," in *Proceedings of the 5th European conference on Computer systems*. ACM, 2010, pp. 363–376.

[3] R. Van Renesse, C. Ho, and N. Schiper, "Byzantine chain replication," in *International Conference On Principles Of Distributed Systems*. Springer, 2012, pp. 345–359.

[4] S. Duan, H. Meling, S. Peisert, and H. Zhang, "Bchain: Byzantine replication with high throughput and embedded reconfiguration," in *International Conference on Principles of Distributed Systems*. Springer, 2014, pp. 91–106.