Ternary Augmented Raft Architecture

Ganesh Prasad Kumble

Abstract

This paper presents a new consensus algorithm based on Byzantine Fault Tolerant[1] RAFT [2] with a hybrid architecture framework that proposes linear as well as exponential robustness in the back-end systems supporting various sorts of transactions that demand speed, resilience, accuracy, integrity and reliability of the node owners.

1 Background

Distributed systems have become quintessential in empowering the digital growth of organizations in many verticals. It is a shared responsibility among ourselves to simplify and provide for an architecture that can exhibit stable properties in hosting financial or commodity transactions from multiple verticals that demand a trusted environment. Readers are highly encouraged to go through the cited articles on PBFT [1], RAFT [2], and Tangaroa [3] thoroughly before reviewing this paper.

2 Introduction

The TARA architecture and the consensus algorithm leverages a compound set of theories such as Practical Byzantine Fault Tolerance[1], Raft Algorithm[2], and cryptographic message exchange etc. on top of the distributed systems[3]. Each concept derived from the above is effectively combined to form an architecture that serves the core purpose of laying out a stable architecture to transact, manage and represent relevant transactions of diverse commodities.

2.1 Basic Network

The TARA architecture comprises of modular components that are supported by the theory of Practical Byzantine Fault Tolerance[1].

As the name of the proposed architecture suggests, each service cluster shall comprise of minimum of 3 nodes. The reason for the cardinality of nodes shall be explained further in accordance to the concept of byzantine failures.

A minimum of 2 service clusters are connected to each other in the form of a distributed bus, connected via an interconnect node called “star node”, hence

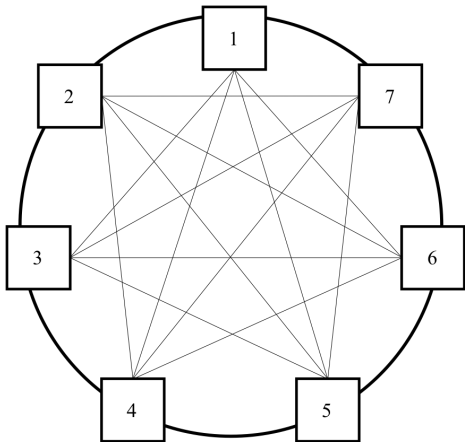

Figure 1: An arbitrary distributed network to demonstrate TARA architecture.

forming a basic operational case of seven nodes in a distributed environment with the presence of one "grand node".

The star node is a randomly elected leader as per the election rules of Raft Algorithm[2], more elaborated in further sections.

2.2 Grand Nodes

Grand nodes are independent group of nodes distributed across the geography to perform the rolse similar to gatekeepers. Gatekeeping activities include the reception of incoming transactions, classify them based on the relevant asset class tagged within the payload. Grand nodes are dedicated resources that possess above the par computational power to perform gatekeeping activities in a reliable manner. Nodes with a dedicated amount of computational hardware, NTQ and stake can participate as a candidate to be elected as one of the grand node. All the above 3 three qualifying factors are dynamic in nature with respect to the underlying network's changes.

2.3 Service Cluster

The service clusters provides response-oriented functionality upon client request. Hence, each node representing a service node may contain service logic(the algorithm) and content(the state) to be delivered upon the client request.

Each service cluster consists of a leader called "A mode" node or "V mode" node based on the nature of the leader in the term. If the leader is performing an output transaction (TX), then it behaves as an V mode node. Else, if the leader is performing an input transaction (RX), then it behaves as a A node. As mentioned in the Raft Algorithm[2], the leader node broadcasts the received message or transaction to its followers in the current term to commit them into respective logs and checks the 'nextIndex' value to determine its nature of

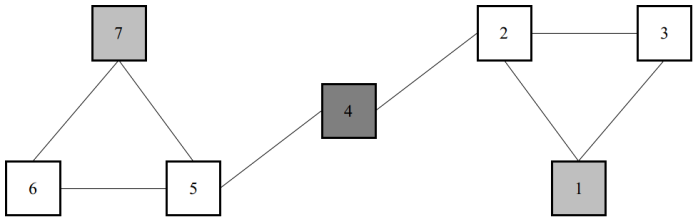

Figure 2: An ideal case of TARA(1,1) network demonstrating TARA architecture for 1 service cluster, each representing A mode leader and V leader respectively. Node 1 is the leader in the A mode cluster and Node 7 is the leader in the V mode cluster. Node 4 is the star node.

operation for the next term. For the sake of simple explanation, we observe the leader to be in either one of the mode. However, the system prescribes to the concept of multi-processing since modern computers are able to equip hybrid nature. If a leader fails to manage the transaction log, a new election shall take place as mentioned in the Raft algorithm[2].

2.4 Star Nodes

Star nodes are the federated leaders of a small cluster within the network, with its size ranging from a minimum of 4 nodes and above. The duty of a Star Node is to recieve the set of grouped transactions by grand nodes and perform formal verification. Majority of the service nodes are expected to pass the formal verification by default to propogate a block. The formally verified transactions are now bundled into a new block based on the timestamp proximity and propogated across other clusters in the network. The election of Star Nodes within the service cluster follows the rules of RAFT leader election.

2.5 Service Nodes

Service Nodes are the followers in the service cluster. The service nodes recieve the proposed transactions from their leader, the Star Node, and perform formal verification of their own.

3 Algorithm

3.1 Federated leadership

3.2 Message passing

3.3 Log replication and consensus

3.4 Follow-up Sync

4 Correctness

4.1 Managing Failures

As proposed in the concept of Practical Byzantine Fault Tolerance [1] and Tangaroa [3] , a BFT Raft cluster designed to tolerate f number of Byzantine failures must contain at least $n = 3f + 1$ nodes, where n - f nodes form a quorum [3].

Consider, to sustain a maximum number of 2 failures, n = 7. Our test network depicted in Fig3.2 matches the requirements by satisfying the value of n in the above equation. Hence, we understand that 5 nodes may form a cluster of trust, or a quorum, to receive and commit client requests, thereby maintaining the nature of distributed systems. However, as per the equation mentioned above, another byzantine / non-byzantine failure will leave the entire system in a confused state on the previous commits that happened in asynchronous flow of requests. Considering the 2 byzantine failures, we obtain the following:

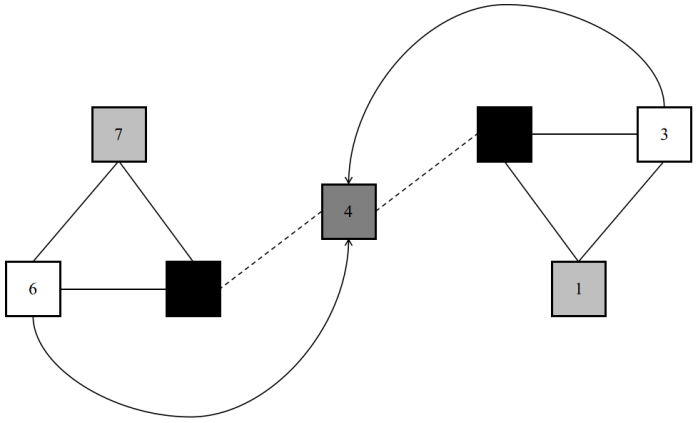

Figure 3: An ideal case of TARA((1,-1)(1,-1)) network demonstrating TARA architecture with each service cluster suffering from one byzantine / non-byzantine fault.

Since a cluster of 3 nodes can tolerate up-to 1/3rd of fault, both service clusters are in-tact. If one more byzantine or non-byzantine fault occurs within each

service cluster, we may observe that the cluster is inconsistent thereby making the entire system unusable in the case of arbitrary non-TARA architectures.

4.2 Resilience

The novel aspect of the TARA architecture can be observed by considering the failure scenario mentioned above. A TARA distributed network suffering from 2 byzantine faults in each service cluster will respond proactively in the following manner:

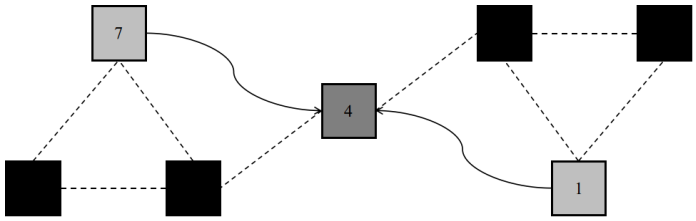

Figure 4: An ideal case of TARA((1,-2)(1,-2)) network demonstrating TARA architecture with each service cluster suffering from two byzantine / non-byzantine faults.

From the figure 3.4, we clearly observe that the remaining node in each ternary augmented cluster connects automatically to the star node for committing its transaction log to the latest previous index. Here, we observe the importance of the star node, the global state machine, comprising of logs from committed logs of leader from all the clusters.

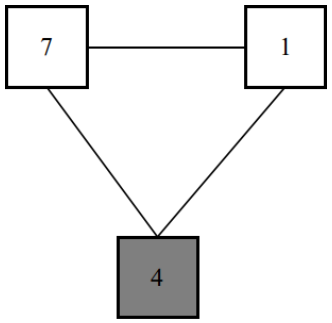

Figure 5: New triad formed as a result of 2 byzantine / non-byzantine failures, TARA((1,-2)(1,-2)).

This change in the network will create a new triad (as shown in the figure 3.5 above), leaded by the star node in the first term of the new triad composed. We may also observe that the newly composed triad is capable of tolerating 1/3rd of fault, i.e., the system is capable of tolerating one more node failure, byzantine or not.

Hence, the proposed model of architecture, is able to tolerate up-to a maximum of n - f failures out of f expected byzantine failures and form a quorum decently up-to f, and hence making the system more usable than any other proposed models in [1] and [3].

5 Applicability

5.1 Sentinel Scalability

5.2 Modulated Trust

5.3 Throughput

6 Implementation and evaluation

6.1 Reference implementation

The implementation of the proposed architecture doesn't require special care or teaching, but needs the practical approach to implementing its technique showcased to drastically reduce the rate of absolute system failure.

We encourage the reader to contact the author, if any correspondence is required to assist/guide in implementing or evaluating the architecture on an existing BFT Raft system fulfilling the design requirements mentioned in Section 3.1.

Since Raft systems communicate through RPCs, we propose the reader to create a new RPC for nodes to communicate the point of failure and current state to the star node in case of TARA((1,-2), (1,-2)). The AppendEntries RPC[2] from leader may be extended to adding the identity of star nodes.

6.2 Performance

7 Related work

Numerous consensus algorithms have been designed, specified and implemented to address a scalable consensus for large scale internetworks such as Blockchain. Some of the algorithms are as follows:

- **Tendermint BFT** The Tendermint BFT consensus algorithm

8 Conclusion

We have proposed TARA, with various design-centric attributes and mathematical techniques that prove the liveness, safety and certainty of the distributed system. The proposed architecture not only prescribes to the theories of Practical Byzantine Fault Tolerance, Raft algorithm etc., but also improves upon the

robustness of the system over failure rates, thereby making the system free of fault-prone environments and increasing availability without added cost of time and compute.

9 Acknowledgement

We are grateful to the authors of all the cited articles, papers and code implementations for providing invaluable knowledge, research resources, time, guidance and feedback on the TARA architecture.

References

[1] M. Castro and B. Liskov, "Practical byzantine fault tolerance," *SDI. Vol. 99*, 1999.

[2] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," 2014.

[3] C. Copeland and H. Zhong, "Tangaroa: a byzantine fault tolerant raft," 2016.