TARA: Ternary Augmented Raft Architecture

Ganesh Prasad Kumble

January 2017

Revision 3

Abstract

TARA (Ternary Augmented Raft Architecture) is an architecture based on the Byzantine Fault Tolerant[1] RAFT Algorithm[2]. The architecture proposes linear as well as exponential robustness in the back-end systems supporting various sorts of transactions that demand absolute speed, resilience, accuracy, integrity and trust.

1. Introduction

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 yet provide architecture that can exhibit stable properties and support financial and commodity transactions from multiple verticals that demand a trusted environment. We highly encourage the reader to go through the cited articles [1], [2], and [3] thoroughly before reviewing this paper.

2. Overview

The TARA architecture leverages a compound set of theories such as Practical Byzantine Fault Tolerance, Raft Algorithm, and cryptographic message exchange etc. on top of the distributed systems. 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 a commodity.

3. Design

3.1 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 exactly 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 forming a basic operational case of seven nodes in a distributed environment. The star node is a randomly elected leader as per the election rules of Raft Algorithm[2], more elaborated in further sections.

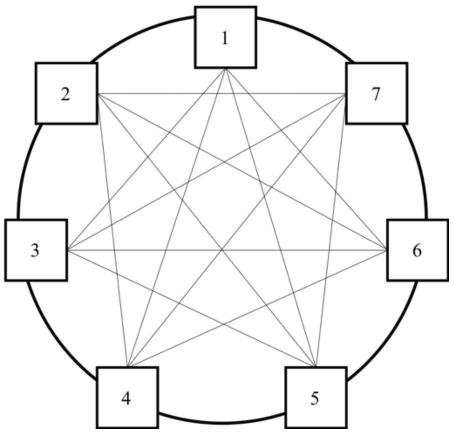

Fig 3.1: An arbitrary distributed network to demonstrate TARA architecture.

3.2 Service clusters

The service clusters provides response-oriented functionality upon client request. Hence, each node representing a service node may contain service logic and service content 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[2] to determine its nature of 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].

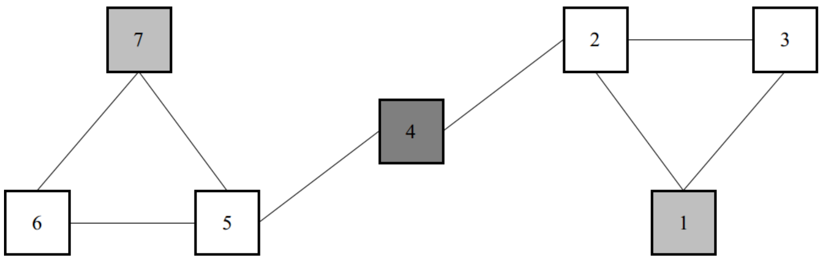

Fig 3.2: An ideal case of TARA(1,1) network demonstrating TARA architecture for 1 service cluster, each representing A mode leader and V leader respectively. Node 1 is the leader in the A mode cluster and Node 7 is the leader in the V mode cluster. Node 4 is the star node.

3.3 Star node

The TARA architecture proposes the election of the star node in a justified manner of balancing the core ideology of preserving the consistency and managing the leadership among the existing cluster leaders. The architecture proposes the leaders of respective clusters to vote for non-leaders of the clusters under the current term. The rules of election follows those proposed in the Raft Algorithm[2].

Unlike the followers of a service node, the star node listens to the leaders from both the service clusters depending upon the case and nature of operation designed, thereby generating a master log replication process that preserves the state-machine data of the entire network. Hence, we may consider the star nodes as collective "*Global State Machine*". The justification for this *'redundancy'* is mentioned further.

3.4 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 $n \geq 3f + 1$ 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:

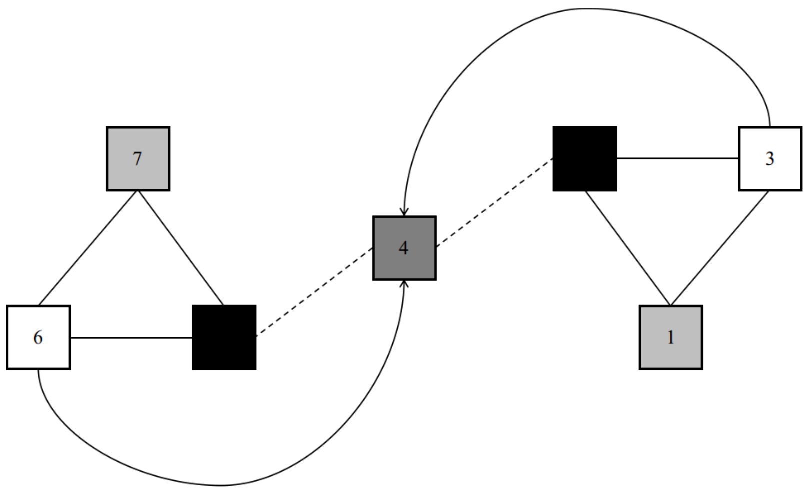

Fig 3.3: An ideal case of TARA((1,-1)(1,-1)) network demonstrating TARA architecture with each service cluster suffering from one byzantine / non-byzantine fault.

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.

3.5 Correctness

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:

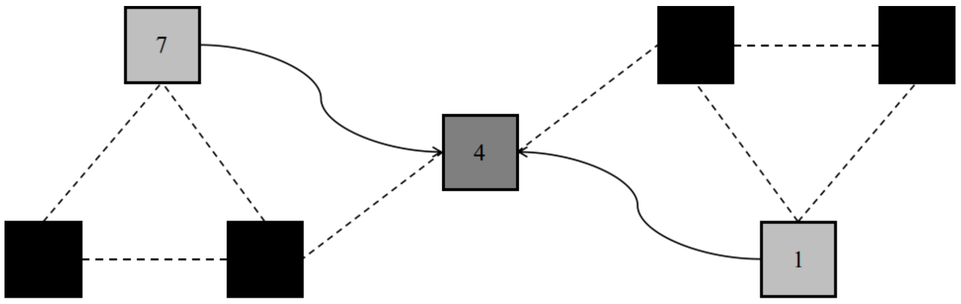

Fig 3.4: An ideal case of TARA((1,-2)(1,-2)) network demonstrating TARA architecture with each service cluster suffering from two byzantine / non-byzantine faults.

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.

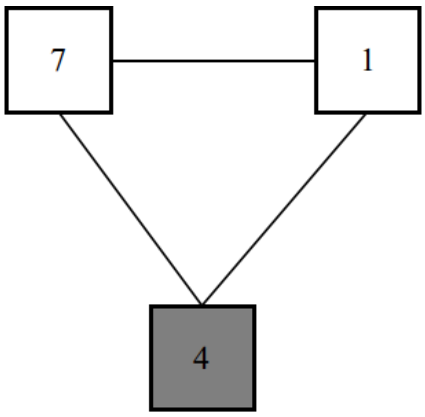

Fig 3.5: New triad formed as a result of 2 byzantine / non-byzantine failures, TARA((1,-2)(1,-2)).

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].

4. Implementation and evaluation

The implementation of the above architecture doesn't require special care or teaching, but the practical approach to implementing the architecture and 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 BFT Raft systems communicate through RPCs (Remote Procedure Call), 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 the star node.

5. Conclusion

We have proposed the TARA architecture, with various design-centric 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.

6. Acknowledgment

The author of the paper is grateful to the authors of cited papers [1], [2] & [3] for providing invaluable knowledge, research resources, time, guidance and feedback on the TARA architecture.

References

[1] Castro, Miguel, and Barbara Liskov. “Practical Byzantine Fault Tolerance.” SDI. Vol. 99. 1999.

[2] Ongaro, Diego, and John Ousterhout. “In Search of an Understandable Consensus Algorithm.” 2014.

[3] Christopher Copeland and Hongxia Zhong. “Tangaroa: a Byzantine Fault Tolerant Raft” 2016.