A Review of 'RBFT: Redundant Byzantine Fault Tolerance'

Prashanth R Duggirala

Summary

Paper by Pierre-Louis Aublin, Sonia Ben Mokhtar and Vivien Quema, 2013

This paper tries to address the issue that most of the robust BFT protocols which were being proposed were not effectively robust. This is mainly due to the fact that they rely on a dedicated replica called primary. This primary can be smartly malicious, that means, despite efforts from other replicas to control that it behaves correctly, it can slow the performance down to the detection threshold, without being caught. So the authors tried to address this by multiple redundant primaries. In RBFT, multiple instances of a BFT protocol are executed in parallel. Each instance has a primary replica. The various primary replicas are all executed on different machines. While all protocol instances order requests, only one instance (called the master instance) effectively executes them. Other instances (called backup instances) order requests in order to compare the throughput they achieve to that achieved by the master instance. If the master instance is slower, the primary of the master instance is considered malicious and the replicas elect a new primary at each protocol instance.

Like most other BFT protocols, RBFT also requires 3f + 1 nodes (i.e., 3f + 1 physical machines). RBFT is actually implemented on the foundations of Aardvark, using it's codebase, while improving upon the performance. Each node runs f + 1 protocol instances of a BFT protocol in parallel. This means that each of the N nodes in the system runs one replica for each protocol instance locally and the different instances order the requests following a 3-phase commit protocol similar to PBFT. To do this RBFT leverages multicore architectures of modern hardware. The replicas always keep monitoring the throughput of the primaries and trigger recovery whenever the primary running the master instance is slow. In RBFT, all the f + 1 instances receive the same client requests. It follows a six step process from a client request to reply back much like PBFT with the Verification, Propagation, Dispatch & Monitoring and Execution steps but it includes one additional propagation step. In the propagation step, when a node receives a client request, it does not give it directly to the f + 1 replicas it is hosting. Rather, it forwards the request to all other nodes. When a node has received 2f+1 copies of a client request (possibly including its own copy), it knows that every correct node will eventually receive the request (because the request has been sent to at least one correct node). Consequently, it gives the request to the f + 1 replicas it hosts. All these steps are implemented as separate modules so that they can be executed on separate threads and the replicas are implemented as separate processes to effectively leverage multiprocessing advantages offered by multicore processor architectures. The paper compares the performance of RBFT against the state of-the-art robust BFT protocols in and show that RBFT has comparable performance to state-of-the-art protocols in the fault-free case. They also analyse the performance of RBFT under some of the possible worst-case attacks and show that the throughput degradation is limited to 3%.

Strong points

1. Tackles the issue of smartly malicious primaries which can slow the performance down to the detection threshold and avoid detection by correct replicas.

2. RBFT leverages multicore architectures to run multiple instances of the same protocol in parallel. Nodes have to compare the throughput achieved by the different instances to know whether a protocol instance change is required or not.

3. RBFT adopts separate Network Interface Controllers (NICs) to avoid client traffic to slow down node-to-node communication and also protect against a flooding attack performed by faulty nodes.

Weak points

1. Huge cryptographic and network overhead because of all the redundancy, even when the instances are processed in parallel. Moreover, they run experiments with only upto two Byzantine faults, i.e., $f \leq 2$.

2. RBFT is only intended for open loop systems, the algorithm's functionality in the closed world case has not been worked upon in this paper.

3. The usage and implementation details of digital signatures + MAC addresses for message forwarding has not been explained.

Comments expanding upon weak points

First, the sheer number of messages that are exchanged between the instant when a client sends a request to the system to the instant where the system sends a reply back is too high because of the $f + 1$ instances of the PBFT protocol running for every request, this works fine until the parallel architecture of the system hardware is utilized to their fullest potential. If the number of nodes increase further, the system cannot scale that well because the number of cores is limited. In their evaluation, they haven't scaled the system to any possible real world scenario, this might be because of the above mentioned reasons. Second, in RBFT protocol, clients may send multiple requests in parallel without waiting for the reply of a request before sending new requests. In a closed loop system, backup instances would never be faster than the master instance, which means RBFT could never make use of the redundant instances running in parallel. Third, the paper does not explain anything about how the message authenticity is being checked, it mentions the usage of digital signatures along with a MAC address based check but I'm not clear on how that is being implemented in this protocol.