Adapting to Evolving Threats against BFT Systems with **Weighted** and **Diverse** Replication

Alysson Bessani
Byzantine Fault Tolerance Protocols

• **Performance**
  • The race horses: PBFT, Zyzzyva, Alyph, ...

• **Robustness**
  • Slow but steady: Prime, Aardvark, RBFT, ...

• **Resource efficiency**
  • Strong assumptions: MinBFT, CheapBFT, XFT, ...

• **Scalability**
  • Blockchainers: HoneyBadger, FastBFT, SBFT, ...
This talk: **Adaptiveness**

- **Changing/Unknown Network Conditions**
  - Weighted Replication applied to BFT
  - Autonomic weight assignment

- **Threat Landscape (attacks and vulnerabilities)**
  - Diversity for fault independence
  - Monitoring new vulnerabilities and changing the replica set
BFT-SMaRt

BACKGROUND


BFT-SMaRt

• Byzantine/Crash fault tolerant state machine replication library
  • Written in Java, maintained and evolved during more than 10 years
• Key features: modularity, reconfigurations, robustness, performance
Mod-SMaRt: Normal Phase

Validation is done here

Proof comprised of these signed messages
Mod-SMaRt: Synchronization Phase

IMPORTANT: It looks like PBFT, but it is not PBFT 😊
Adaptivity to Network Conditions

PART I


Classical BFT Replication

N=4, f=1

Egalitarian quorums,
Any 3 out-of 4 replica
Experimental study conducted with BFT-SMaRt on Planetlab and Amazon EC2

**Summary:**
- Leader in the best-connected site yields better results than employing a rotating or multiple leader(s) strategy
- Smaller quorums create opportunities for improving latency
**WHEAT**: WeigHt-Enabled Active replicaTion

- Use optimizations that lead to significant latency reduction:
  - Single leader in the best-connected site
  - Tentative executions (from PBFT)
  - Employs smaller quorums (weighted replication)

- Weighted replication: safe voting assignment scheme for SMR
  - Uses $\Delta$ extra replica(s) for quorum formation
  - Improves latency by enabling more choice upon quorum formation
  - Needs $a$ to preserve quorum intersection and tolerance to $f$ faulty replicas
Weighted BFT Replication

N=4, f=1, Δ=1 (extra)

Weighted quorums, One set of 3 out-of 5 and any set 4 out-of 5
Weighted BFT Replication

• **Consistency**: All quorums that hold Q votes intersect by at least one correct replica

• **Availability**: There is always a quorum available in the system that holds Q votes

• **Safe minimality**: There exists at least one minimal quorum in the system
Weighted BFT Replication

Define the number of replicas $u$ that hold $V_{\text{max}} > 1$ votes, without violating $f$

<table>
<thead>
<tr>
<th>CFT mode</th>
<th>BFT mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n = 2f + 1 + \Delta$</td>
<td>$n = 3f + 1 + \Delta$</td>
</tr>
<tr>
<td>$N_v = \sum V_i = 2F_v + 1$</td>
<td>$N_v = \sum V_i = 3F_v + 1$</td>
</tr>
<tr>
<td>$Q_v = F_v + 1$</td>
<td>$Q_v = 2F_v + 1$</td>
</tr>
<tr>
<td>$u = f$</td>
<td>$u = 2f$</td>
</tr>
</tbody>
</table>

Input: $f$ and $\Delta$

Output: $u$ and $V_{\text{max}}$

$V_{\text{max}} = \frac{\Delta + f}{f} = 1 + \frac{\Delta}{f}$
Size of fast quorums with different $f$ and $\Delta$
**AWARE**: Adaptive Wide-Area REplication

The benefit of weighted replication depends on **choosing an optimal weight configuration**

- The environment of the system (i.e., network characteristics) may **change at runtime** (e.g., due to a DDoS attack)

AWARE enables a geo-replicated consensus-based system to **adapt to its environment**!
AWARE Approach

• **Self-Monitoring**
  • AWARE uses reliable self-monitoring as decision-making basis for adapting replicas’ voting weights and leader position at runtime

• **Self-Optimization**
  • AWARE continuously strives for consensus latency gains at runtime
  • Changes weights and leader location to minimize consensus latency
Self-Monitoring: Measuring Latency

• Each replica measures its point-to-point latency to other replicas for consensus protocol messages

• **Non-Leader’s Propose**
  • Periodically an alternately selected dummy leader broadcasts a dummy proposal

• **Write-Response**
  • Replicas immediately respond by sending acknowledgments
Self-Monitoring: Consolidating Measurements

• Replicas periodically disseminate their measurements to others with **total order** until they have the same latency matrices

• AWARE maintains **synchronized matrices** for both PROPOSE and WRITE latencies $\hat{M}^p$ and $\hat{M}^w$ used for decisions later

<table>
<thead>
<tr>
<th></th>
<th>Oregon</th>
<th>Ireland</th>
<th>Sydney</th>
<th>Sao Paulo</th>
<th>Virginia</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oregon</td>
<td>0</td>
<td>65</td>
<td>69</td>
<td>92</td>
<td>40</td>
</tr>
<tr>
<td>Ireland</td>
<td>65</td>
<td>0</td>
<td>132</td>
<td>93</td>
<td>38</td>
</tr>
<tr>
<td>Sydney</td>
<td>69</td>
<td>132</td>
<td>0</td>
<td>158</td>
<td>105</td>
</tr>
<tr>
<td>Sao Paulo</td>
<td>92</td>
<td>93</td>
<td>158</td>
<td>0</td>
<td>61</td>
</tr>
<tr>
<td>Virginia</td>
<td>40</td>
<td>38</td>
<td>105</td>
<td>61</td>
<td>0</td>
</tr>
</tbody>
</table>
Self-Optimization

• With the same matrices $\hat{M}_P$ and $\hat{M}_W$ the replicas can **solve deterministically** the following optimization problem:

$$\langle \hat{l}, \hat{W} \rangle = \arg\min_{W \in \mathcal{W}, l \in \mathcal{L}} \text{PredictLatency}(l, W, \hat{M}_P, \hat{M}_W)$$

• All correct replicas reach the same, optimal weight distribution and invoke a reconfiguration in the system
Evaluation of WHEAT and AWARE
Setup

- **AWARE** is implemented on top of WHEAT, which is based on BFT-SMaRt.
- For evaluation, we use the Amazon AWS cloud, using EC2 instances of t2.micro type with 1 vCPU, 1 GB RAM and 8 GB SSD volume.
- We select regions **Oregon**, **Ireland**, **Sydney**, **São Paulo** and **Virginia** for instances (1 client and 1 replica on each instance).
- Clients simultaneously send 1kB-requests across all sites.
Clients’ Request Latency

Average latencies of all clients and 20 configurations

Observations

• The best configuration (<4,0>) performs about 39% faster than the median (<3,4>), 64% faster than the worst (<2,1>)
• Tuning voting weights can reduce latency (see configs. with the same leader)
• Leader relocation may be necessary for achieving optimal consensus latency
Figure 8 also shows the clients' observed request latency. As expected, consensus speed contributes to total latency. This is the reason the clients' observed average request latencies are a reasonable for average clients' request latency of $L^\text{max}$. We notice a positive correlation between our series (over all configurations) of model predictions for leader consensus latency. We deploy AWARE in our usual setting and observe its behavior during the system's lifespan. Overall, the clients' observed request latency and the measured latencies are between our series (over all configurations) of model predictions for leader consensus latency. In our experiment, the measured latencies are varied. Variations, we argue that these results are optimal candidates for some configuration that is close to the optimum. We induce events to evaluate AWARE's reaction: AWARE redistributes the replicas' request timers expire and BFT-SMaRt reaction: AWARE attributes one of the replicas, AWARE creates network perturbations, in particular, we add an outgoing delay of $200 \text{ ms}$, and the worst is $521 \text{ ms}$. If there is an optimal configuration, we start AWARE in a low-performance configuration and the measurement series becomes slow. While $V_{\text{max}}$ of Ireland becomes fast again, the network stabilizes and the communication links of Ireland are between $30 \text{ ms}$ and $120 \text{ ms}$, and the measurement series. According to its reaction, clients observe faster request latencies identical to what happened after the first reconfiguration (Event 2). Reaction: AWARE notices this improvement and assigns a new leader. Action: We create network perturbations, in particular, we add an outgoing delay of $200 \text{ ms}$, and the measurement series becomes fast again. Reaction: AWARE attributes one of the remaining correct replicas is forced to use the replica, AWARE decides that a small improvement in request latencies. Reaction: AWARE redistributes the replicas, AWARE creates network perturbations, in particular, we add an outgoing delay of $200 \text{ ms}$.
Adapting to Evolving Threats with Diverse Replication

PART II

Problem

“We assume independent node failures.”

“We assume independent node failures.”

“We assume independent node failures.”

“We assume independent node failures.”

“We assume independent node failures.”
LAZARUS

• LAZARUS is a response to the long-standing open problem of managing the diversity of a BFT system to make it resilient to malicious (intelligent) adversaries

• LAZARUS objective is to address two types of threats:
  • Newly found vulnerabilities affecting one or more active replicas
  • Zero-day vulnerabilities affecting multiple running replicas
LAZARUS Overview

UNTRUSTED

Execution plane

R0 R1 R2 Rn

LTU

BFT agreement

Controller

TRUSTED

Small local trusted component that reboots the host

Logically-centralized orchestrator for reconfigurations

Not controlled by the adversary

Clients

BFT-Replicated service

OSINT

CVE Details

EXPLOIT DATABASE

NVD
Diversity-aware Reconfigurations

1. Find common vulnerabilities
   • Fetch OSINT from NVD, ExploitDB, vendor sites, etc.
   • For each vulnerability, discover
     • exploit and patch information
     • which replicas in the pool are affected

2. Measure vulnerability severity

3. Measure configuration risk
   • Sum the severity of vulnerabilities affecting each pair of replicas from a config.

4. Select next configuration
   • The one with less risk that requires less changes
LAZARUS Implementation

Centralized controller (can be made BFT)

Replicas packed as VirtualBox VMs
Effectiveness and Performance of LAZARUS
LAZARUS Effectiveness

- Using a knowledge base of OS vulnerabilities (NVD, ExploitDB, etc.) from 2014 to the month before the execution (in 2018)
- We consider a run compromised if a published vulnerability affects more than one host from the configuration
- Each strategy was executed a thousand times
Diversity and Performance

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>#Cores</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>UB14</td>
<td>Ubuntu 14.04</td>
<td>4</td>
<td>15GB</td>
</tr>
<tr>
<td>UB16</td>
<td>Ubuntu 16.04</td>
<td>4</td>
<td>15GB</td>
</tr>
<tr>
<td>UB17</td>
<td>Ubuntu 17.04</td>
<td>4</td>
<td>15GB</td>
</tr>
<tr>
<td>OS42</td>
<td>OpenSuse 42.1</td>
<td>4</td>
<td>15GB</td>
</tr>
<tr>
<td>FE24</td>
<td>Fedora 24</td>
<td>4</td>
<td>15GB</td>
</tr>
<tr>
<td>FE25</td>
<td>Fedora 25</td>
<td>4</td>
<td>15GB</td>
</tr>
<tr>
<td>FE26</td>
<td>Fedora 26</td>
<td>4</td>
<td>15GB</td>
</tr>
<tr>
<td>DE7</td>
<td>Debian 7</td>
<td>4</td>
<td>15GB</td>
</tr>
<tr>
<td>DE8</td>
<td>Debian 8</td>
<td>4</td>
<td>15GB</td>
</tr>
<tr>
<td>W10</td>
<td>Windows 10</td>
<td>4</td>
<td>1GB</td>
</tr>
<tr>
<td>WS12</td>
<td>Win. Server 2012</td>
<td>4</td>
<td>1GB</td>
</tr>
<tr>
<td>FB10</td>
<td>FreeBSD 10</td>
<td>4</td>
<td>1GB</td>
</tr>
<tr>
<td>FB11</td>
<td>FreeBSD 11</td>
<td>4</td>
<td>1GB</td>
</tr>
<tr>
<td>SO10</td>
<td>Solaris 10</td>
<td>1</td>
<td>1GB</td>
</tr>
<tr>
<td>SO11</td>
<td>Solaris 11</td>
<td>1</td>
<td>1GB</td>
</tr>
<tr>
<td>OB60</td>
<td>OpenBSD 6.0</td>
<td>1</td>
<td>1GB</td>
</tr>
<tr>
<td>OB61</td>
<td>OpenBSD 6.1</td>
<td>1</td>
<td>1GB</td>
</tr>
</tbody>
</table>
Conclusions

• Distributed trust → geo-distributed systems
• Geo-distributed on the wild → heterogeneity
• Heterogeneity (i.e., diversity) is also important for fault independence

• Our works are “initial steps” on dealing with heterogeneity:
  • Weighted replication is a powerful tool, and other schemes are possible
  • Self-adaptiveness of protocols will become mandatory in operation
  • BFT diversity is a hard problem that need to be addressed
Questions?

- Alysson Bessani
  - anbessani@fc.ul.pt
  - www.di.fc.ul.pt/~bessani

- To know more:
  - BFT-SMaRt & BFT Fabric Orderer: https://github.com/bft-smart/