Reducing the Costs of Large-Scale BFT Replication

Marco Serafini & Neeraj Suri

TU Darmstadt, Germany
BFT state machine replication

- Potential **holy grail** of reliable distributed computing
- Can be used to make **any** deterministic application tolerant to **worst case** failures
- Replication is **transparent** to clients and applications
Generality = Optimality

Generality of BFT requires:
- Minimizing performance overhead and replication costs...
- ... for the widest range of scenarios / workloads

Goal: Identify a general (= optimal) solution for a general problem

Generic application
Single-server implementation

Generic client
Sees a single reliable server

BFT replication library
Toward optimal BFT Replication

- Much work on the topic
  - PBFT [OSDI’99] – Use MACs instead of signatures
  - Q/U [SOSP’05] – Reduce latency using quorum systems
  - FaB [TDSC’06] – Fast agreement
  - Zyzzyva [SOSP’07] – Speculation

Diagram:
- Generic client
  - Sees a single reliable server
- BFT replication library
- Generic application
  - Single-server implementation

© Marco Serafini
State of the Art

- The search for the holy grail has done a long way
- Still, it is not yet over

<table>
<thead>
<tr>
<th></th>
<th>Replication costs</th>
<th>Optimal Fault-free</th>
<th>Optimal w. faults</th>
<th>Strengthen for attacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBFT</td>
<td>$3f + 1$</td>
<td>NO</td>
<td>NO</td>
<td>YES*</td>
</tr>
<tr>
<td>Zyzzyva</td>
<td>$3f + 1$</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Zyzzyva5</td>
<td>$5f + 1$</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Example: Web Applications

- Ideal setting for applying BFT
  - Exposed to the Internet, strong reliability requirements
Web Applications’ Requirements

- Large scale: Benign faults are the common case

- High performance for ALL requests
  - Example: Dynamo’s SLA specifies worst-case latency for 99.9% of the requests under high load [SOSP’07]
  - ALL = in presence of (benign) failures

- Low replication costs
  - 100s to 1,000s of replicated services
  - Additional replication costs must be multiplied over the number of services
Web Applications and BFT Facts

- Existing approaches are **not optimal**
  - **PBFT**: Poor throughput with web application workload
    - BFS has 45% throughput reduction for replicated vs. non-replicated BFS using the Postmark benchmark [TOCS’02]
Web Applications and BFT Facts

- Existing approaches are **not optimal**
  - **PBFT**: Poor throughput with web application workload
    - BFS has 45% throughput reduction for replicated vs. non-replicated BFS using a the Postmark benchmark [TOCS’02]
  - **Zyzzyva**: Improves on PBFT in fault-free runs but has performance degradation in faulty runs
    - Up to 2x higher throughput than PBFT in fault-free runs [SOSP07]
    - Up to 35% throughput reduction with benign failures [SOSP’07]
Web Applications and BFT Facts

- **Existing approaches are not optimal**
  - **PBFT**: Poor throughput with web application workload
    - BFS has 45% throughput reduction for replicated vs. non-replicated BFS using a the Postmark benchmark [TOCS’02]
  - **Zyzzyva**: Improves on PBFT in fault-free runs but has performance degradation in faulty runs
    - Up to 2x higher throughput than PBFT in fault-free runs [SOSP07]
    - Up to 35% throughput reduction with benign failures [SOSP’07]
  - **Zyzzyva5**: Improves on PBFT also in faulty runs but has 50% higher replication costs [SOSP’07]
Web Applications and BFT Facts

- Existing approaches are **not optimal**
  - **PBFT**: Poor throughput with web application workload
    - BFS has 45% throughput reduction for replicated vs. non-replicated BFS using a the Postmark benchmark [TOCS’02]
  - **Zyzzyva**: Improves on PBFT in fault-free runs but has performance degradation in faulty runs
    - Up to 2x higher throughput than PBFT in fault-free runs [SOSP07]
    - Up to 35% throughput reduction with benign failures [SOSP’07]
  - **Zyzzyva5**: Improves on PBFT also in faulty runs but has 50% higher replication costs [SOSP’07]

- Can we improve on this?
**Contribution: The Scrooge Protocol**

- Optimal solution for common applications
  - Tolerate 1 Byzantine fault (+ crashes)

<table>
<thead>
<tr>
<th></th>
<th>Replication costs</th>
<th>Optimal fault-free</th>
<th>Optimal benign faults</th>
<th>Strengthen for attacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBFT</td>
<td>$3f + 1$</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>Zyzzyva</td>
<td>$3f + 1$</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Zyzzyva5</td>
<td>$5f + 1$</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Scrooge</td>
<td>$4f$</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>
Seminal work on reducing the costs of BFT

- **Optimal** resilience
- **Three** phases: **Non-optimal**
- $O(n^2)$ message complexity: **Non-optimal**
Optimized for common, *speculative* runs

Speculative replies contain *history digests*

- Clients can check that all correct replicas are consistent before delivery → no explicit agreement is required

Optimal latency for consensus

\(O(n)\) message complexity
In *non-speculative* runs

- Execute second phase for all subsequent requests
- Client acts as a relay to complete it
- Remove the third phase
What is a Speculative Run?

- Definition depends on the redundancy used
What is a Speculative Run?

- Definition depends on the redundancy used

- With $3f+1$ replicas: *Fault-free* runs
  - Benign clients
  - *Fault-free* replicas
What is a Speculative Run?

- Definition depends on the redundancy used

- With $3f+1$ replicas: *Fault-free* runs
  - Benign clients
  - *Fault-free* replicas

- With $5f+1$ replicas: *Faulty* runs
  - Benign clients
  - Correct primary and *faulty* backups
Scrooge Contribution

- Speculation in *faulty* runs with $4f$ replicas
  - Optimal resilience for common applications
Scrooge Contribution

- Speculation in *faulty* runs with 4f replicas
  - Optimal resilience for common applications

- Two novel ideas
Scrooge Contribution

- Speculation in *faulty* runs with $4f$ replicas
  - Optimal resilience for common applications

- Two novel ideas
  1. Explicitly identify a *Repliers Quorum*
Scrooge Contribution

- Speculation in faulty runs with 4f replicas
  - Optimal resilience for common applications

- Two novel ideas
  1. Explicitly identify a Repliers Quorum
  2. Backups store whole order request messages from the primary in their history
Initially identify a **Repliers Quorum** of $N-f$ replicas.
Initially identify a Repliers Quorum of $N-f$ replicas

1. **Primary orders the requests**
Initially identify a Repliers Quorum of $N-f$ replicas

1. Primary orders the request

2. Backups store the **whole message** in the history
Initially identify a Repliers Quorum of $N-f$ replicas

1. Primary orders the request
2. Backups store the whole message in the history
3. Only replicas in the **Repliers Quorum** reply
Initially identify a Repliers Quorum of $N-f$ replicas

1. Primary orders the request
2. Backups store the whole message in the history
3. Only replicas in the Repliers Quorum reply
4. Clients deliver after receiving $N-f$ replies from $RQ$
Existing Lower Bounds

- Additional replicas are necessary for fast agreement in all faulty runs
Existing Lower Bounds

- Additional replicas are necessary for fast agreement in all faulty runs

- Scrooge: No additional replicas in common applications & fast agreement in faulty runs
Existing Lower Bounds

- Additional replicas are necessary for fast agreement in all faulty runs

- Scrooge: No additional replicas in common applications & fast agreement in faulty runs

- Why is Scrooge consistent with the lower bounds?
  - Eventually re-establish speculation upon failure events
  - Detect and isolate faults, no speculation in the meanwhile - for a bounded time
  - System model: All BFT protocols use MACs – leverage this
Strengthening BFT

- **Aardvark**: PBFT live under attacks
  - Periodically change primary and estimate throughput
  - Use few alternative communication patterns

- **Why Zyzzyva is not suitable**
  - No third phase, replicas cannot observe progress
  - Multiple alternative patterns if clients are faulty

- **Scrooge does not have these limitations**
  - Can be strengthened similar to PBFT

Conclusions

- BFT protocols must be **optimal** to represent a truly generic technique for dependability
- Scrooge reaches optimality for common applications where $f = 1$

<table>
<thead>
<tr>
<th></th>
<th>Replication costs</th>
<th>Optimal fault-free</th>
<th>Optimal w. faults</th>
<th>Strengthen for attacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scrooge</td>
<td>$4f$</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

- Scrooge opens up new issues
  - Scrooge is an upper bound. Does it represent a **lower bound** too?
  - Can we have a more sophisticated failure detection?
Thank you for your attention