Prophecy: Using History for High-Throughput Fault Tolerance

Siddhartha Sen

Joint work with Wyatt Lloyd and Mike Freedman

Princeton University
Non-crash failures happen
Non-crash failures happen

Model as Byzantine (malicious)
Mask Byzantine faults
Mask Byzantine faults

Throughput

Clients

Replicated service
Mask Byzantine faults

Throughput

Clients

Replicated service

facebook

facebook

facebook

facebook
Mask Byzantine faults

Throughput

Clients

Replicated service
Mask Byzantine faults

Throughput

Clients

Replicated service

facebook

facebook

facebook

facebook
Mask Byzantine faults

Clients

Throughput

Linearizability (strong consistency)

Replicated service
Byzantine fault tolerance (BFT)

- Low throughput
- Modifies clients
- Long-lived sessions
Prophecy

• High throughput + good consistency

• No free lunch:
  – Read-mostly workloads
  – Slightly weakened consistency
Byzantine fault tolerance (BFT)

• Low throughput
  - D-Prophecy

• Modifies clients
  - Prophecy

• Long-lived sessions
Traditional BFT reads

Clients

Replica Group

application
Traditional BFT reads

Clients → Agree? → Replica Group

application
A cache solution

Clients

cache

application

Replica Group
A cache solution

Clients

Agree?

Cache

Application

Replica Group
A cache solution

Problems:
- Huge cache
- Invalidation

Clients

Cache Replica Group

Application
A compact cache

<table>
<thead>
<tr>
<th>Requests</th>
<th>Responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>req1</td>
<td>resp1</td>
</tr>
<tr>
<td>req2</td>
<td>resp2</td>
</tr>
<tr>
<td>req3</td>
<td>resp3</td>
</tr>
<tr>
<td>⋮</td>
<td>⋮</td>
</tr>
</tbody>
</table>
A compact cache

<table>
<thead>
<tr>
<th>Requests</th>
<th>Responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>sketch(req1)</td>
<td>sketch(resp1)</td>
</tr>
<tr>
<td>sketch(req2)</td>
<td>sketch(resp2)</td>
</tr>
<tr>
<td>sketch(req3)</td>
<td>sketch(resp3)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
A sketcher

Clients

Replica Group

sketcher
application
Executing a read

Clients

Replica Group
 Executing a read

Clients

sketch

webpage

Replica Group
Executing a read

Clients

Replica Group

sketch

webpage
Executing a read

Agree? 😊

Clients

sketch

webpage

Replica Group
Executing a read

Fast, load-balanced reads
Executing a read

Agree?

Clients

Replica Group

sketch

webpage
Executing a read

Clients

Replica Group

sketch

webpage
Executing a read

Clients

key-value store

replicated state machine

Replica Group

sketch

webpage
Executing a read

Clients

Replica Group

sketch

webpage
Executing a read

Clients

Replica Group

sketch

webpage
Executing a read

Clients

Replica Group

sketch

webpage
Executing a read

Agree?

Clients

Replica Group

sketch

webpage
Executing a read

Maintain a fresh cache

Agree? 😊

Clients

Replica Group

sketch

webpage
Did we achieve linearizability?

NO!
Executing a read

Clients

Replica Group

sketch

webpage

……

facebook
Executing a read

Clients

Replica Group

sketch

webpage

facebook
Executing a read

Agree? 😞

Clients

Replica Group

sketch

webpage

facebook
Executing a read

Clients

sketch

webpage

Replica Group
Executing a read

Agree?

Clients

sketch

webpage

Replica Group

Clients

Replica Group
Executing a read

Fast reads may be stale

Clients

Agree?

Replica Group

Facebook

sketch

webpage
Load balancing

Clients

Replica Group

sketch

webpage

facebook
Load balancing

Agree?

Clients

Replica Group

sketch

webpage

facebook
Load balancing

Pr($k$ stale) = $g^k$

Clients

Replica Group

Agree?

sketch

webpage

facebook

……

……

……

……
D-Prophecy vs. BFT

Traditional BFT:
• Each replica executes read
• Linearizability

D-Prophecy:
• One replica executes read
• “Delay-once” linearizability
Byzantine fault tolerance (BFT)

- Low throughput
- Modifies clients
- Long-lived sessions

D-Prophecy

Prophecy
Key-exchange overhead

![Graph showing throughput vs. session length for PBFT-ro]
Key-exchange overhead

![Graph showing throughput vs session length for PBFT-ro with 11% overhead.]
Key-exchange overhead

- Exchange overhead: 11%
- 3%
Internet services

Clients

Replica Group
A proxy solution

Clients

Proxy

Replica Group
A proxy solution

Consolidate sketchers

Clients

Proxy

Replica Group
A proxy solution

Consolidate sketchers

Clients

Sketcher

Replica Group
A proxy solution

Sketcher must be fail-stop

Clients

Sketcher

Trusted

Replica Group
A proxy solution

- Trust middlebox already
- Small and simple

Clients ➔ Sketcher ➔ Trusted ➔ Replica Group
Executing a read

Clients → Sketcher (Trusted) → Replica Group
Executing a read

Clients

Sketcher

Replica Group
Executing a read

Clients

Replica Group

Sketched

Trusted
Executing a read

<table>
<thead>
<tr>
<th>Req</th>
<th>Resp</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s(q)$</td>
<td>x</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Executing a read

Clients

Sketcher

Replica Group
Executing a read

Clients

Sketcher

Replica Group
Executing a read

Clients

Replica Group

Sketched

Trusted
Executing a read

Clients

Sketch

Req | Resp
---|---
$s(q)$ | |
| | |

Replica Group
Executing a read

Clients

Sketch

trusted

Replica Group

<table>
<thead>
<tr>
<th>Req</th>
<th>Resp</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s(q)$</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Prophecy

Clients

Sketcher

Trusted

Replica Group
Prophecy

Fast, load-balanced reads

Clients

Sketcher

Replica Group
Prophecy

Fast reads may be stale

Clients

Sketcher

<table>
<thead>
<tr>
<th>Req</th>
<th>Resp</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s(q) )</td>
<td></td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
</tr>
</tbody>
</table>

Replica Group

facebook
Delay-once linearizability
Delay once linearizability
Delay once linearizability

\[ \langle W, R, W, W, R, R, W, R \rangle \]
Delay once linearizability

Read-after-write property

\[ \langle W, R, W, W, R, R, W, R \rangle \]
Delay-once linearizability

\[ \langle W, R, W, W, R, R, W, R \rangle \]
Example application

• Upload embarrassing photos
  1. Remove colleagues from ACL
  2. Upload photos
  3. (Refresh)

• Weak may reorder

• Delay-once preserves order
Byzantine fault tolerance (BFT)

- Low throughput
- Modifies clients
- Long-lived sessions

D-Prophecy

Prophecy
Implementation

• Modified PBFT
  – PBFT is stable, complete
  – Competitive with Zyzzyva et. al.

• C++, Tamer async I/O
  – Sketcher: ~2000 LOC
  – PBFT library: ~1140 LOC
  – PBFT client: ~1000 LOC
Evaluation

- Prophecy vs. proxied-PBFT
  - Proxied systems

- D-Prophecy vs. PBFT
  - Non-proxied systems
Evaluation

- Prophecy vs. proxied-PBFT
  - Proxied systems

- We will study:
  - Performance on “null” workloads
  - Performance with real replicated service
  - Where system bottlenecks, how to scale
Basic setup

Clients (100)  

Sketcher  
(concurrent)  

Replica Group (PBFT)
Fraction of failed fast reads
Fraction of failed Alexa top sites: < 15%  

Fraction of failed fast reads
Small benefit on null reads

![Graph showing throughput vs. transition ratio with two lines, one labeled Prophecy and the other pr-PBFT-ro. The graph indicates a small benefit on null reads.](image-url)
Small benefit on null reads
Apache webserver setup

Clients → Sketcher → Replica Group

Apache webserver setup
Large benefit on real workload
Large benefit on real workload

![Graph showing throughput vs. transition ratio for different protocols, with a large benefit indicated at 3.7x]
Large benefit on real workload

Throughput (Kreqs/s) vs Transition Ratio

- Prophecy
- pr-PBFT-ro

- 3.7x
- 2.0x
Large benefit on real workload

Throughput (Kreqs/s)

Transition Ratio

Prophecy
pr-PBFT-ro

3.7x
2.0x
Benefit grows with work

![Graph showing normalized throughput vs. processing time]
Benefit grows with work
Benefit grows with work
Benefit grows with work

94μs (Apache)
Benefit grows with work

94μs (Apache)

Null workloads are misleading!

Normalized Throughput

Prophecy

pr-PBFT-ro

Processing Time (μs)
Benefit grows with work
Single sketcher bottlenecks

![Graph showing normalized throughput vs. response size (KB) for Prophecy, Prophecy-15, and pr-PBFT-ro. The graph indicates a decrease in throughput as the response size increases.]
Single sketcher bottlenecks

![Graph showing normalized throughput vs response size (KB) for different protocols: Prophecy, Prophecy-15, and pr-PBFT-ro. The graph indicates a decrease in throughput as response size increases.]
Scaling out
Scales linearly with replicas

![Graph showing throughput vs. replica group size]
Summary

• Prophecy good for Internet services
  – Fast, load-balanced reads

• D-Prophecy good for traditional services

• Prophecy scales linearly while PBFT stays flat

• Limitations:
  – Read-mostly workloads (meas. study corroborates)
  – Delay-once linearizability (useful for many apps)
Thank You
Additional slides
Transitions

• Prophecy good for read-mostly workloads

• Are transitions rare in practice?
Measurement study

- Alexa top sites

- Access main page every 20 sec for 24 hrs
Mostly static content
Mostly static content
Mostly static content

15%
Dynamic content

• Rabin fingerprinting on transitions

• 43% differ by single contiguous change

• Sampled 4000 of them, over half due to:
  – Load balancing directives
  – Random IDs in links, function parameters