Making Byzantine Fault Tolerant Systems Tolerate Byzantine Failures

Allen Clement, Mirco Marchetti, Edmund Wong Lorenzo Alvisi, Mike Dahlin
BFT Systems

- PBFT [OSDI 98]
- HQ [OSDI 06]
- Zyzzyva [SOSP 07]
- HT BFT [DSN 04]
- QU [SOSP 05]
- BFT Under Attack [NSDI 08]
- Commit Barrier Scheduling [SOSP 07]
- Low Overhead BFT [SOSP 07]
- Attested Append Only Memory [SOSP 07]
- Beyond 1/3 Faulty in BFT [SOSP 07]
- BASE [OSDI 02]
- SafeStore [USENIX 07]
- Separating Agreement from Execution [SOSP 03]
- SUNDR [OSDI 04]
- ...
## System Throughput

<table>
<thead>
<tr>
<th></th>
<th>Best Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBFT</td>
<td>62k</td>
</tr>
<tr>
<td>Q/U</td>
<td>24k</td>
</tr>
<tr>
<td>HQ</td>
<td>15k</td>
</tr>
<tr>
<td>Zyzzyva</td>
<td>65k</td>
</tr>
</tbody>
</table>

ops/sec
## System Throughput

<table>
<thead>
<tr>
<th></th>
<th>Best Case</th>
<th>Faulty Client</th>
<th>Client Flood</th>
<th>Faulty Primary</th>
<th>Faulty Replica</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PBFT</strong></td>
<td>62k</td>
<td>0</td>
<td>crash</td>
<td>1K</td>
<td>250</td>
</tr>
<tr>
<td><strong>Q/U</strong></td>
<td>24k</td>
<td>0</td>
<td>crash</td>
<td>NA</td>
<td>19k</td>
</tr>
<tr>
<td><strong>HQ</strong></td>
<td>15k</td>
<td>NA</td>
<td>4.5k</td>
<td>NA</td>
<td>crash</td>
</tr>
<tr>
<td><strong>Zyzzyva</strong></td>
<td>65k</td>
<td>0</td>
<td>crash</td>
<td>crash</td>
<td>0</td>
</tr>
</tbody>
</table>

**ops/sec**
## System Throughput

<table>
<thead>
<tr>
<th></th>
<th>Best Case</th>
<th>Faulty Client</th>
<th>Client Flood</th>
<th>Faulty Primary</th>
<th>Faulty Replica</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PBFT</strong></td>
<td>62k</td>
<td>0</td>
<td>crash</td>
<td>1K</td>
<td>250</td>
</tr>
<tr>
<td><strong>Q/U</strong></td>
<td>24k</td>
<td>0</td>
<td>crash</td>
<td>NA</td>
<td>19k</td>
</tr>
<tr>
<td><strong>HQ</strong></td>
<td>15k</td>
<td>NA</td>
<td>4.5k</td>
<td>NA</td>
<td>crash</td>
</tr>
<tr>
<td><strong>Zyzzyva</strong></td>
<td>65k</td>
<td>0</td>
<td>crash</td>
<td>crash</td>
<td>0</td>
</tr>
<tr>
<td><strong>Aardvark</strong></td>
<td>39k</td>
<td>39k</td>
<td>7.8k</td>
<td>37k</td>
<td>11k</td>
</tr>
</tbody>
</table>

*ops/sec*
Outline

- Robust BFT: The case for a new goal
- Aardvark: Designing for RBFT
- Evaluation: RBFT in action
Paved with good intentions

- No BFT protocol should rely on synchrony for safety.

- FLP: No consensus protocol can be both safe and live in an asynchronous system!
  - All one can guarantee is eventual progress.

- “Handle normal and worst case separately as a rule, because the requirements for the two are quite different: the normal case must be fast; the worst case must make some progress.”
  -- Butler Lampson, “Hints for Computer System Design”
Recasting the problem

- Maximize performance when
  - the network is synchronous
  - all clients and servers behave correctly

- While remaining
  - safe if at most $f$ servers fails
  - eventually live
Recasting the problem

- Misguided
- Dangerous
- Futile
Recasting the problem

- Misguided
  - it encourages systems that fail to deliver BFT
- Dangerous
- Futile
Recasting the problem

- Misguided
  - it encourages systems that fail to deliver BFT

- Dangerous
  - it encourages fragile optimizations

- Futile
Recasting the problem

- Misguided
  - it encourages systems that fail to deliver BFT

- Dangerous
  - it encourages fragile optimizations

- Futile
  - it yields diminishing return on common case
A New Goal

Asynchronous

Failures

Synchronous

No Failures
A New Goal

Asynchronous

Failures

Synchronous

Failures

Synchronous

No Failures
A New Goal

<table>
<thead>
<tr>
<th>Asynchronous</th>
<th>Synchronous</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failures</td>
<td>Failures</td>
</tr>
</tbody>
</table>
Robust BFT

- Maximize performance when
  - the network is synchronous
  - at most $f$ servers fail
- While remaining
  - safe if at most $f$ servers fail
  - eventually live
Outline

- Robust BFT: The case for a new goal
- Aardvark: Designing for RBFT
- Evaluation: RBFT in action
Protocol Structure

- **Step 1**: "Good" messages
- **Step 2**: "Bad" messages
- **Step 3**: Computation steps

- "Good" messages
- "Bad" messages
Fragile Optimizations
Revisiting conventional wisdom

- Signatures are expensive - use MACs
- View changes are to be avoided
- Hardware multicast is a boon
Revisiting conventional wisdom

- Signatures are expensive - use MACs
  - Faulty clients can use MACs to generate ambiguity
  - Aardvark requires clients to sign requests
- View changes are to be avoided
- Hardware multicast is a boon
Revisiting conventional wisdom

- Signatures are expensive – use MACs
  - Faulty clients can use MACs to generate ambiguity
    - Aardvark requires clients to sign requests

- View changes are to be avoided
  - Aardvark uses regular view changes to maintain high throughput despite faulty primaries

- Hardware multicast is a boon
Revisiting conventional wisdom

- Signatures are expensive - use MACs
  - Faulty clients can use MACs to generate ambiguity
    - Aardvark requires clients to sign requests

- View changes are to be avoided
  - Aardvark uses regular view changes to maintain high throughput despite faulty primaries

- Hardware multicast is a boon
  - Aardvark uses separate work queues for clients and individual replicas
Big MAC Attack

\[ C \]

- 
- 
- 
- 
-
Big MAC Attack
Big MAC Attack
Big MAC Attack

Diagram with labeled nodes and connections.
Big MAC Attack
Big MAC Attack
Big MAC Attack
Big MAC Attack
Big MAC Attack
Big MAC Attack

[Diagram showing a relationship between different elements labeled with 'C' and a symbol.]
Big MAC Attack
Big MAC Attack

Faulty Client

Faulty Primary
Hybrid MAC/Signatures
Hybrid MAC/Signatures

[Diagram showing hybrid MAC/Signatures with request and submission]

[Diagram description: Hybrid MAC/Signatures diagram showing the flow of request and submission with different colored nodes and lines.]
Hybrid MAC/Signatures

The MAC is good. How is the signature?
Hybrid MAC/Signatures

C

request submission
Hybrid MAC/Signatures

Signature is good too!
Hybrid MAC/Signatures

request

submission
Hybrid MAC/Signatures

request submission
primary orders request
Hybrid MAC/Signatures
Hybrid MAC/Signatures

- request submission
- primary orders
- request

Diagram:

1. Initial state
2. Event or action
3. Request
4. Submission
5. Primary orders
6. Request

Diagram shows the flow of events and dependencies.
Signed Request Filtering

Client
Blacklisted?

Verify
MAC

Verify
Signature

Blacklist
Client

Process
Request
Big MAC Attack

PBFT

request submission  primary orders request  replicas agree on the next request  replicas respond to the client
Big MAC Attack

Zyzzyva

- request submission
- primary orders request
- execute the request
- replicas agree on the next request
- replicas respond to the client
Big MAC Attack

- Request submission
- "Primary" orders request
- Replicas agree on the next request
- View change
- Replicas respond to the client
Big MAC Attack

request submission
replicas agree on the next request
view change
replicas respond to the client
execute the request
Slow Primary

- [ ]
- [ ]
- [ ]
- [ ]
- [ ]
Slow Primary
Slow Primary
Adaptive View Changes

Throughput

Time

Observed Throughput

Required Throughput
Adaptive View Changes

Time

Throughput

Observed Throughput

Required Throughput
Adaptive View Changes

Throughput

Time

- Observed Throughput
- Required Throughput
Adaptive View Changes

Throughput

Time

Observed Throughput

Required Throughput
Implementation details

- Sign client requests
- Adaptive view change
- Separate network channels
- Fair scheduling
- clients -v- replicas
- replicas -v- replicas
- Exploit multicore architectures
Outline

- Robust BFT: The case for a new goal
- Aardvark: Designing for RBFT
- Evaluation: RBFT can work
Throughput -v- Latency
Aardvark, Incrementally

<table>
<thead>
<tr>
<th></th>
<th>MAC Client Request</th>
<th>Sign Client Request</th>
<th>Adaptive View Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBFT</td>
<td>62k</td>
<td>30k</td>
<td>–</td>
</tr>
<tr>
<td>Aardvark</td>
<td>58k</td>
<td>39k</td>
<td>39k</td>
</tr>
</tbody>
</table>
Performance with failures

- Byzantine failures are arbitrary
- Good faith effort
## Big MAC Attack

<table>
<thead>
<tr>
<th></th>
<th>Peak</th>
<th>Faulty Client</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBFT</td>
<td>62k</td>
<td>0</td>
</tr>
<tr>
<td>Q/U</td>
<td>24k</td>
<td>0</td>
</tr>
<tr>
<td>HQ</td>
<td>7.6k</td>
<td>-</td>
</tr>
<tr>
<td>Zyzzyva</td>
<td>65k</td>
<td>0</td>
</tr>
<tr>
<td>Aardvark</td>
<td>39k</td>
<td>39k</td>
</tr>
</tbody>
</table>
## Slow Primary

<table>
<thead>
<tr>
<th></th>
<th>Peak</th>
<th>1ms delay</th>
<th>10ms delay</th>
<th>100ms delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBFT</td>
<td>62k</td>
<td>5k</td>
<td>5k</td>
<td>1k</td>
</tr>
<tr>
<td>Zyzzyva</td>
<td>65k</td>
<td>28k</td>
<td>5k</td>
<td>crash</td>
</tr>
<tr>
<td>Aardvark</td>
<td>39k</td>
<td>38k</td>
<td>37k</td>
<td>38k</td>
</tr>
</tbody>
</table>
Summary

- RBFT: a new goal for BFT systems
- Aardvark: rejecting conventional wisdom
- Evaluation: it works!