CS 138: Practical Byzantine Consensus
Scenario

- Asynchronous system
- Signed messages
- Servers are state machines
- It has to be practical
The Request

Client

Server 0

pre-prepare

Server 1

pre-prepare

Server 2

pre-prepare

Server 3

pre-prepare

Server 0

Request

Client
Non-Primaries Respond (1)
Non-Primaries Respond (2)

Client

Server 0

Server 1

prepare

prepare

prepare

Server 3

Server 2

Client
Non-Primaries Respond (3)

Client

Server 0

prepare

Server 1

prepare

Server 2

prepare

Server 3

prepare

Client
Servers Commit to Request (1)
Servers Commit to Request (2)

Client

Server 0

Server 1

commit

Server 2

commit

Server 3

commit

Client
Servers Commit to Request (3)
Servers Commit to Request (4)

Client

Server 0

Server 1

commit

Server 2

Client

Server 3
All Respond to Client

- Client
  - Server 0
  - Server 1
  - Server 2
  - Server 3
Contents of Messages

\[\text{pre-prepare: seq #; digest(msg)} \quad \text{msg}\]

\[\text{prepare: seq #; digest(msg), i}\]
Be Prepared

• n servers, at most $\text{floor}((n-1)/3)$ faulty servers
• A non-primary server is prepared when
  – it has received pre-prepare message
  – it has received matching prepare messages from $2f-1$ other non-primaries
    - $2f$ other non-primaries including itself
• It’s prepared to believe the primary
  – both content of request and request sequence
However ...

- There are multiple clients, each sending a sequence of requests
- Communication isn’t perfect
  - messages may arrive out of order
- Server s may be prepared, but s’ is not
  - but will be eventually
- Server may be prepared for request q but not for q-1
Commitment

- Server $i$ multicasts *commit* message to all others when it is prepared

  
  \[
  \textit{commit}: \text{ seq #; digest(msg), i}
  \]

- A message is *committed* if it is prepared at $f+1$ non-faulty servers
  - how does an individual server know this?
    - it is prepared and has received $2f$ commits from others

- Server executes message when
  - message is committed
  - and all previous messages have been executed
Logging

• Each server maintains log of
  – pre-prepares
  – prepares
  – commits
Checkpoints

• Checkpoint = state of replica after all messages through a particular sequence number have been executed

• Log can be trimmed when all agree on replicas’ states

• Servers periodically exchange signed checkpoint messages
  – contain digest of checkpoint

• Checkpoint messages from 2f+1 different servers constitute a proof of the checkpoint

• Log up to the checkpoint can be replaced with checkpoint and its proof
Traitorous Primary

- Client sends request
- No response from primary
- Client re-sends request to all servers
- Servers forward request to primary
- If no response, then need new primary
Views

• A particular primary server is in charge of a view $v$

• If the primary changes, the view changes to $v+1$
  – the primary for view $v$ is server $v \mod S$
    - $S$ is the number of servers
View Changes (1)

• Non-primaries who time-out waiting for server send signed *view-change* messages
  – provide
    - most recent checkpoint plus proof
    - list of prepared messages since checkpoint
  • with proof: pre-prepare plus prepare messages
View Changes (2)

- New primary, after receiving $2f$ valid view-change messages, responds with new-view message
  - provides
    - set of view-change messages
      - i.e., proof of view change
    - list of pre-prepare messages for all prepared messages since checkpoint
      - missing messages are nullified
  - non-primaries move to new view and reprocess prepared messages in this view
Performance

• BFS: Byzantine fault-tolerant NFS
  – replicated NFS servers
  – simplified implementation of NFS
    - NFSv2

• Implementations tested
  – BFS: 4 servers
  – BFS-nr: one server
  – NFS-std: Digital Unix NFSv2
Andrew Benchmark

- phase 1
  - creates subdirectories recursively
- phase 2
  - copies a source-code tree
- phase 3
  - examines status of all files without reading their data
- phase 4
  - reads all data bytes
- phase 5
  - compiles and links all files
## BFS vs BFS-nr

<table>
<thead>
<tr>
<th>phase</th>
<th>strict</th>
<th>r/o lookup</th>
<th>BFS-nr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.55 (57%)</td>
<td>0.47 (34%)</td>
<td>0.35</td>
</tr>
<tr>
<td>2</td>
<td>9.24 (82%)</td>
<td>7.91 (56%)</td>
<td>5.08</td>
</tr>
<tr>
<td>3</td>
<td>7.24 (18%)</td>
<td>6.45 (6%)</td>
<td>6.11</td>
</tr>
<tr>
<td>4</td>
<td>8.77 (18%)</td>
<td>7.87 (6%)</td>
<td>7.41</td>
</tr>
<tr>
<td>5</td>
<td>38.68 (20%)</td>
<td>38.38 (19%)</td>
<td>32.12</td>
</tr>
<tr>
<td>total</td>
<td>64.48 (26%)</td>
<td>61.07 (20%)</td>
<td>51.07</td>
</tr>
</tbody>
</table>
## BFS vs. NFS

<table>
<thead>
<tr>
<th>phase</th>
<th>BFS</th>
<th>NFS-std</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>strict</td>
<td>r/o lookup</td>
</tr>
<tr>
<td>1</td>
<td>0.55 (-69%)</td>
<td>0.47 (-73%)</td>
</tr>
<tr>
<td>2</td>
<td>9.24 (-2%)</td>
<td>7.91 (-16%)</td>
</tr>
<tr>
<td>3</td>
<td>7.24 (35%)</td>
<td>6.45 (20%)</td>
</tr>
<tr>
<td>4</td>
<td>8.77 (32%)</td>
<td>7.87 (19%)</td>
</tr>
<tr>
<td>5</td>
<td>38.68 (-2%)</td>
<td>38.38 (-2%)</td>
</tr>
<tr>
<td>total</td>
<td>64.48 (3%)</td>
<td>61.07 (-2%)</td>
</tr>
</tbody>
</table>