Attested Append-only Memory: Making Adversaries Stick to their Word

Distributed Storage Systems
CS 6464
2-19-09

presented by: Hussam Abu-Libdeh
Motivation

- You want to build a service
  - Easy on a single machine
  - What about failure and reliability?
    - Replicate service on multiple machines
- Replicated services must appear as a single server
  - Linearizability
    - Completed client requests appear to have been processed in a **single, totally ordered, serial schedule** consistent with the order they were submitted
Motivation

• Machines can fail or be hijacked
  • Byzantine failure
    – Can not distinguish if node is non-faulty, faulty, or malicious

• Faulty servers can lie
  • Equivocation
    – Different lies to different people

• Previously in cs6464, SUNDR & fork consistency
Today

Can we use small trusted components to combat equivocation?
Agenda

• Equivocation “attacks”
• The A2M
• A2M-PBFT-E
• A2M-PBFT-EA
• A2M-Storage
• A2M-PBFT-EAXYZ-FOO-RANDOM-CHARS
  • Ok maybe not
• Discussion
Equivocation

• Servers respond incorrectly and differently to different clients
  • Can be detected if clients were trusted
• Could happen in two places
  • Servers equivocating to clients
  • Servers equivocating to other servers
• Both bad
Equivocating to Clients
Equivocating to Servers
A2M

- Attested Append-only Memory
- A trust abstraction
- Essentially:
  - A chunk of memory
  - You can access it
  - You trust its content
    - You have a reason to trust it
    - Backed up by a TPM, or placed in a trusted VM or VMM or on a separate trusted machine ..etc
A2M Interface

• Supports basic operations
  • append(q,x)
    – Add value to the tail of the list
  • lookup(q,n,z)
    – Look up value at position n
  • end(q,z)
    – Look up last entry in list
  • truncate(q,n)
    – Remove all entries below n
  • advance(q,n,d,x)
    – Skip a few positions (n-current position) in the list
PBFT

- Practical Byzantine Fault-Tolerance
- Client sends request, later a reply is accepted if received from more than 1/3 of the servers
- Internally works in 3 phases
  - Primary multicasts `pre-prepare` to all replicas
  - If a server receives `pre-prepares` from > 2/3 of the servers, it multicasts a `prepare` message
  - If a server receives `prepares` from > 2/3 of the servers, it multicasts a `commit` message
PBFT
PBFT

• Two steps of PBFT
  • Agreement
    − pre-prepare, prepare, and commit messages
  • Execution
    − communication between replicas and client

• Other parts of PBFT
  • Checkpointing, changing views ..etc
  • Not central to our discussion today
A2M-PBFT-E

- PBFT with trusted Execution step (A2M)
  - Replicas can equivocate to each other
  - Equivocation to clients will be detected
- Clients accept reply quorums if all agree in A2M entry for the reply sequence number
  - Requires > 2/3 replicas be non-faulty (like PBFT)
  - If 1/3 < # faulty < 2/3
    - Clients won't commit faulty sequence #s because at least one replica will have correct A2M entry
A2M-PBFT-E

(b) A2M-PBFT-E
A2M-PBFT-EA

- PBFT w/ trusted Execution & Agreement steps
  - Equivocation to clients will be detected
  - Equivocation to servers will be detected
- At each step, replicas attest msgs with A2M
  - Just need a majority (>1/2) of replicas to agree
  - Thus can tolerate <1/2 of faulty servers
A2M-PBFT-EA
A2M-Storage

- Server maintains two A2M logs
  - One for operation digest (like SUNDR) (log s)
  - One for latest write sequence number (log h)
- Client use timestamps with read/write
  - Timestamp = (req_seq, att_seq_h, att_seq_s)
    - i.e. Client operations attest if current sequence # is latest
  - Clients store their last timestamp
- Read/Write operations use timestamp
  - If latest, proceed; otherwise, refresh
Evaluation

- Emulated A2M in a C++ module
- Ran agreement protocol w/ 4 replicas & 1 client
- Microbenchmark
  - requests/replies of various size
- Macrobenchmark
  - NFS front-end with PBFT backend
  - Compile a relatively small package
- Results not surprising
Evaluation - Microbenchmarks

![Graphs showing processing time vs. request size and response size for different protocols.](image-url)
## Evaluation - Macrobenchmarks

<table>
<thead>
<tr>
<th>Phase</th>
<th>NFS</th>
<th>-S</th>
<th>-PBFT</th>
<th>-A2M-PBFT-E (sig)</th>
<th>-A2M-PBFT-E (MAC)</th>
<th>-A2M-PBFT-EA (sig)</th>
<th>-A2M-PBFT-EA (MAC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy</td>
<td>0.219</td>
<td>0.709</td>
<td>1.026</td>
<td>0.728</td>
<td></td>
<td>2.141</td>
<td>0.763</td>
</tr>
<tr>
<td>Uncompress</td>
<td>1.015</td>
<td>3.027</td>
<td>4.378</td>
<td>3.103</td>
<td></td>
<td>8.601</td>
<td>3.236</td>
</tr>
<tr>
<td>Untar</td>
<td>2.322</td>
<td>4.448</td>
<td>6.826</td>
<td>4.553</td>
<td></td>
<td>12.896</td>
<td>4.669</td>
</tr>
<tr>
<td>Clean</td>
<td>0.180</td>
<td>0.298</td>
<td>0.640</td>
<td>0.312</td>
<td>0.742</td>
<td>0.311</td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>23.725</td>
<td>28.355</td>
<td>41.821</td>
<td>28.854</td>
<td>61.940</td>
<td>29.528</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Mean time to complete the six macrobenchmark phases in seconds.
Evaluation – Varying delay time

<table>
<thead>
<tr>
<th>Additional latency ($\mu s$)</th>
<th>NFS- A2M-PBFT-E (MAC)</th>
<th>A2M-PBFT-E (MAC) with batching</th>
<th>A2M-PBFT-EA (MAC)</th>
<th>A2M-PBFT-EA (MAC) with batching</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>28.854</td>
<td>28.763</td>
<td>29.528</td>
<td>29.505</td>
</tr>
<tr>
<td>10</td>
<td>29.598</td>
<td>29.025</td>
<td>31.299</td>
<td>30.188</td>
</tr>
<tr>
<td>50</td>
<td>32.735</td>
<td>30.232</td>
<td>36.242</td>
<td>32.214</td>
</tr>
<tr>
<td>250</td>
<td>48.784</td>
<td>37.237</td>
<td>66.441</td>
<td>45.199</td>
</tr>
<tr>
<td>1000</td>
<td>117.59</td>
<td>65.813</td>
<td>192.53</td>
<td>101.62</td>
</tr>
</tbody>
</table>

Table 2: Mean time to complete the six macronbenchmark phases in seconds for different A2M additional latency costs.
Thank You