BAR Fault Tolerance

BAR Fault Tolerance for Cooperative Services

Amitanand S. Aiyer, Lorenzo Alvisi, Allen Clement, Mike Dahlin, Jean-Philippe Martin, and Carl Porth
U.T. Austin CS Dept.

SOSP’05, Oct 2005, Brighton, UK.

*Goal is to develop reliable distributed apps in hostile environment*
System Model

- System spans Multiple Administrative Domains, therefore nodes can be:
  - Byzantine – unpredictable (fewer than \((n-2)/3\) [LSP82])
  - Altruistic – obedient
  - Rational – optimally lazy and self-interested
    - conservative – never risk losing service.
    - non-collusive
    - follow protocol in case of utility “tie”
  - Synchronous – node slowness is unacceptable
  - Expensive, closed membership (no Sybil attack). Real-world sanctions.
  - Unforgeable digital signatures (RSA)
Intuition

- Political and Game theory – Hobbes and Nash
- Criminal justice – punishment as deterrence

Result is BAR-B, a distributed backup service. Deviation from the protocol is always detectable, so any rational node will follow the protocol to prevent exclusion from the service. Reliability measures:

- Incentive-Compatible Byzantine Fault Tolerant (IC-BFT)
- Byzantine Altruistic Rational Tolerant (BART)
Layered approach

- Abstraction of the lower layers allow simplified analysis of the upper.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Prototype</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 3: Application</td>
<td>BAR–B Backup</td>
</tr>
<tr>
<td>Level 2: Work Assignment</td>
<td>Guaranteed Response</td>
</tr>
<tr>
<td></td>
<td>Periodic Work</td>
</tr>
<tr>
<td>Level 1: Primitives</td>
<td>Authoritative Time</td>
</tr>
<tr>
<td></td>
<td>Replicated State Machine</td>
</tr>
<tr>
<td></td>
<td>Message Queue</td>
</tr>
</tbody>
</table>
Layer 1: Reliable State Machine

- Terminating Reliable Broadcast – variant of consensus
  - must satisfy: Termination, Agreement, Integrity, Non-triviality
  - Implemented as three-phase-commit (agree/write/show-quorum)

- Message queue
  - Local retaliation policy – ignore a node until it responds to all of your outstanding requests.
  - enforces *predictable communication patterns*

- Balanced messages
- Penance required of *untimely* nodes
- Garbage collection of *badlist* (slow) nodes
- Global punishment – *Proof Of Misbehavior* required
Layer 2: Enforcement

Transactions are public.

- *Witness* node is implemented in underlying RSM
- Requests and responses are sent through the witness node (broadcasted)
- Deterministic RSM time is used for judging timeouts
- *Fast path* can be used if no trouble expected
- Witness node checks that *periodic work* is completed
Layer 3: The Application

To be IC-BFT, must provide:

- long-term benefit for participation
- reliable backup
- fault tolerance
- erasure coding
- only a small number of recoveries are allowed
- POM verification – *most difficult*
  - message timestamping, signed responses serve as evidence
  - global audit process reviews evidence
- meaningful sanctions
- node exclusion
Experimental setup

- Laboratory deployment
- Emulab used to emulate different network conditions
- Tens of nodes, hundreds of MB data per node
- Reasonable parameters were drawn from a hat:
  - TRB timeout of 10 seconds
  - 40 Guaranteed Response slots
  - 1 month data lease
  - max_response_time of 1 week
  - 3 recoveries (plus one every two years) allowed
Results

Figure 5: RSM performance as nodes are added

Figure 6: Impact of rotating leadership

Figure 7: Operation time for 100 MB

Figure 8: Operation time for 20MB at various encodings
Results (cont.)

Figure 9: Concurrent operations

Figure 10: Operation under different network conditions

Figure 11: Impact of the fast path optimization

Figure 12: Cost of audit as capacity grows
Future work

- Relax model assumptions: collusion, allow risk-taking
- Scalability by partitioning
- More sophisticated punishment schemes
- Analysis of parameter choice
Problems

- Proofs omitted and unintuitive:
  - Byzantine fault tolerance
  - Lots of details
- No measurement of effects of deviant behavior
- Real-world experimental evidence is needed to support user rationality assumption.
- Scalability – witness implementation requires bcast.
- Principle of *cost balancing* is wasteful
- and as always ... confusing use of acronyms