Graceful Degradation through Time Travel

Lidong Zhou
Vijayan Prabhakaran
Rama Ramasubramanian
Roy Levin
Chandu Thekkath

Microsoft Research Silicon Valley
Fault Tolerance vs. Graceful Degradation

- Fault Tolerance
- Graceful Degradation

Graph showing availability vs. failures with usual fault tolerance and fault tolerance + graceful degradation.
Graceful Degradation via Time Travel

• critical to maintain “correctness”
• time travel
• coherent state from the past
  - coherent degraded state

• already used for failure recovery, archival systems, historical databases
Contributions

• three “correctness” specs for graceful degradation

• mechanisms to implement the specs in a versioned storage system

• discussion of tradeoffs between the three specs
Notations and Definitions

- **State Machine (M)**
  - State $M(C_1, C_2, ..., C_n)$

- **Command (c)**
  - Query: does not change state
  - Update: changes state
    - Read Set (c.rs): set of objects read
    - Write Set (c.ws): set of objects written

- **Serialization**

- **Linearization**
Naïve Approach and Problem

- Versioned storage system:
- Time travel: go back in time before failure
  - may lead to invalid state!

<table>
<thead>
<tr>
<th>Command</th>
<th>( C_1 )</th>
<th>( C_2 )</th>
<th>( C_3 )</th>
<th>( C_4 )</th>
<th>( C_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>State</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>\xmark</td>
</tr>
</tbody>
</table>
Spec 1: Prefix Linearizability

- **Coherent Degraded State 1:**

  any state \( M(c_1, c_2, ..., c_k) \) for \( 1 \leq k \leq n \)
  
  for a given execution \((c_1, c_2, ..., c_n)\)
Spec 2: Prefix Serializability

<table>
<thead>
<tr>
<th>Command</th>
<th>( C_1 )</th>
<th>( C_2 )</th>
<th>( C_3 )</th>
<th>( C_4 )</th>
<th>( C_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td></td>
<td>[\text{Read} ]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write</td>
<td></td>
<td></td>
<td>[\text{Write} ]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>State</td>
<td></td>
<td></td>
<td></td>
<td>[\text{State} ]</td>
<td></td>
</tr>
</tbody>
</table>

- **Coherent Degraded State 2:** \[\text{Read} \land \text{Write} \]

any state \( M(c_{i_1}, c_{i_2}, ..., c_{i_k}) \) for \( 1 \leq k \leq n \)
for a valid serialization \( (c_{i_1}, c_{i_2}, ..., c_{i_n}) \)
of a given execution \( (c_1, c_2, ..., c_n) \)
Spec 3: Subsequence Linearizability

- **Coherent Sub-Sequence:**

  a sub-sequence $c_{j_1}, c_{j_2}, ..., c_{j_k}$ for $1 \leq j_1 < j_2 < ... < j_k \leq n$, where value of each object $o \in c_{j_i}.rs \cup c_{j_i}.ws$ is same before execution of $c_{j_i}$ in sub-sequence and the given sequence $(c_1, c_2, ..., c_n)$
Spec 3: Subsequence Linearizability

- **Coherent Degraded State 3:**  
  any state $M(c_{j_1}, c_{j_2}, ..., c_{j_k})$ for $1 \leq j_1 < j_2 < ... < j_k \leq n$  
  for a coherent sub-sequence $(c_{j_1}, c_{j_2}, ..., c_{j_k})$  
  of a given execution $(c_1, c_2, ..., c_n)$
Degree of Degradation

• $n - k$:
  - $n$ is the total number of commands
  - $k$ is the number of commands reflected in the degraded state

• Examples:
  - Prefix linearizability: $4 (c_1)$
  - Prefix serializability: $3 (c_1, c_3)$
  - Subsequence linearizability: $2 (c_1, c_3, c_5)$
Putting it Together

• Normal Mode
  – all required objects are available!
  – command executed atomically (transaction style)

• Degraded Mode (excessive failures)
  – one or more required objects is not available!
  – find a **most recent coherent degraded state** based on a chosen spec
  – execute query on the coherent degraded state
  – abort updates in the degraded mode!
Most Recent Coherent Degraded State

- **State:**
  - a set of versions one for each object

- **Coherent Degraded State:**
  - valid degraded state defined by one of the specs

- **More Recent Coherent Degraded State:**
  - $S_1$ is more recent than $S_2$ if for each object $o$ and versions $<o,v_1> \in S_1$ and $<o,v_2> \in S_2$, $v_1 \geq v_2$

- **Most Recent Coherent Degraded State:**
Spec 1: Timestamp-Based Mechanism

• Timestamps with each version:
  – Write timestamp: \(<o,v>\).wt
    • timestamps are transactional!

• Timestamp-Based Coherent Set \(S\):
  – iff there is a timestamp \(t\) such that:
    for each \(<o,v> \in S,\)
    \(<o,v>\).wt \(\leq t < <o,v>\).succ.wt
Spec 1: Timestamp-Based Mechanism

- Finding the most recent coherent set:
  - Start with a set $S$ of most recent available versions
  - Let $<o,v> \in S$ have the highest timestamp $t$
  - If $\exists <o',v'> \in S$ with $t \geq (o',v').\text{succ.wt}$:
    Replace $<o,v>$ with highest available version of $o$ and timestamp $< t$
Spec 3: Weaker Dependency-Based Mechanism

- **Dependencies with each version:**
  \[
  \forall <o,v> \in c.ws, \quad <o,v>.wdep := c.ws \\
  \quad \cup <ro,v>.wdep \quad \forall <ro,v> \in c.rs \\
  \quad \cup <wo,v>.prev.wdep \quad \forall <wo,v> \in c.ws
  \]

- **Weak-Dependency-Based Coherent Set S:**
  - iff <o_1,v_1> and <o_2,v_2> in S:
    \[
    \text{if } <o_1,v> \in <o_2,v_2>.wdep \Rightarrow v \leq v_1, \\
    <o_2,v> \in <o_1,v_1>.wdep \Rightarrow v \leq v_2
    \]
Spec 3: Weaker Dependency-Based Mechanism

- Finding the most recent coherent set:
  - Start with a set $S$ of most recent available versions
  - If $\exists <o_1,v_1> \in S$, $<o_2,v_2> \in S$, and $<o_1,v> \in <o_2,v_2>.wdep$ with $v > v_1$:
    - Replace $<o_2,v_2>$ with the highest available version of $o_2 < v_2$
Spec 2: Dependency-Based Mechanism

- Dependencies with each version:
  \[ \forall <o,v> \in c.ws, <o,v>.dep := c.ws \]
  \[ \cup <ro,v>.dep \quad \forall <ro,v> \in c.rs \]
  \[ \cup <wo,v>.prev.dep \quad \forall <wo,v> \in c.ws \]
  \[ \cup <wo,v>.prev.equiv \quad \forall <wo,v> \in c.ws \]

  \[ \forall <o,v> \in c.rs, <o,v>.equiv := <o,v>.equiv \]
  \[ \cup <ro,v>.dep \quad \forall <ro,v> \in c.rs \]
  \[ \cup <wo,v>.prev.dep \quad \forall <wo,v> \in c.ws \]
Mechanisms vs. Specs

- Timestamp-based mechanism $\iff$ prefix linearizability (spec 1)
- Dependency-based mechanism $\iff$ prefix serializability (spec 2)
- Weak-Dependency-based mechanism $\iff$ subsequence linearizability (spec 3)
- Proofs in the paper!
Implications

• Timestamp-Based Coherency:
  – accurate time travel to consistent past state
  – Similar to Immortal Databases [LBMSWZ05, SIGMOD]

• Dependency-Based Coherency:
  – virtual time travel
  – degraded state may not have really existed!

• Weaker Dependency-Based Coherency:
  – undo the effects of some commands

Example:
Source-Code tree with independent packages

• Jan 07:
  – Perl, Python

• Feb 07:
  – Python Update

• Mar 07:
  – Perl Update

• Apr 07:
  – Stats Query

• May 07:
  – Python Update
Tradeoffs

• Strength:
  – Spec 1 (PL) ≥ Spec 2 (PS) ≥ Spec 3 (SL)

• Degree of Degradation:
  – Spec 1 (PL) ≥ Spec 2 (PS) ≥ Spec 3 (SL)

• Overhead:
  – Spec 1 (PL) ≤ Spec 3 (SL) ≤ Spec 2 (PS)
Graceful Degradation for Updates

• Degraded updates are hard!

• Permanent failures:
  – perform updates on the degraded state

• Transient failures:
  – branched versions
    • pick latest branch as valid and proceed
    • use a merge operation to combine branches
Summary and Conclusions

- graceful degradation improves availability in the face of excessive failures
- coherency needs to be preserved during degradation
- time travel helps coherent degradation
  - definitions, mechanisms, analysis