Exploring Trade-offs in Transactional Parallel Data Movement

Ivo Jimenez, Carlos Maltzahn (UCSC)
Jay Lofstead (Sandia National Labs)

November 18, 2013
The need for Transactional Atomicity
The difference with Databases

- In terms of ACID, we want:
  - Atomicity
  - Durability
  - Leave Isolation/Consistency to the clients
- Single Transaction (vs. thousands)
- Massive amount of cohorts (vs. hundreds)
The approach

• Assume that storage servers can do:
  • multi-version concurrency control
  • per-object visibility control

• Clients handle consensus
Consensus Protocols

- **NBTA**
  - Initiate: initiate 
  - Solicit votes:
    - canCommit?: yes / no 
  - Commit / abort:
    - commit / rollback 
    - acknowledge 
  - Finalize commit:
    - commit / rollback 
    - acknowledge 

- **2PC**
  - Initiate: initiate 
  - Solicit votes:
    - canCommit?: yes / no 
  - Commit / abort:
    - commit / rollback 
    - acknowledge 
  - Finalize commit:
    - commit / rollback 
    - acknowledge 

- **3PC**
  - Initiate: initiate 
  - Solicit votes:
    - canCommit?: yes / no 
  - Commit / abort:
    - preCommit 
    - acknowledge 
  - Finalize commit:
    - commit / rollback 
    - acknowledge 

- **Paxos Commit**
  - Propose:
    - propose 
    - accept / reject 
  - Prepare:
    - prepare 
    - acknowledge 
  - Finalize commit:
    - commit 
    - acknowledge
NBTA

- **Non-blocking Transactional Atomicity**
- “HAT” formalization (Bailis et al. VLDB 2014)
- In the context of Highly-available systems
- Can also be applied in synchronous systems to achieve very low overhead
Features

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Fault Model</th>
<th>Block</th>
<th>Async</th>
<th>Replication</th>
</tr>
</thead>
<tbody>
<tr>
<td>NBTA</td>
<td>none</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>2PC</td>
<td>fail-stop</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>3PC</td>
<td>fail-stop</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Paxos</td>
<td>fail-recover</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Our goal

• One-size-fits-all solution won’t work
• Let users pick based on their needs:
  • Length of job
  • MTTF
  • fault modes
  • etc
• We want to explore trade-offs and characterize protocols based on the user needs
Preliminary Evaluation

NBTA vs. 2PC

Time (microseconds, log-scale)

Number of Processes

- NBTA
- 2PC
Future Work

• Incorporate fault-tolerance
  • Cohort failure: can recover individually
  • Coordinator failure: 3PC, Paxos

• Coordinate asynchronously
  • No need to wait for global consensus
Related Work

• DOE’s Fast Forward Storage and I/O. The FastForward approach is similar to the NBTA protocol.
• Fault-tolerant MPI make use of consensus protocols to identify faulty processes.
• Recovery in multi-level checkpoint restart.
Thanks!