Distributed and Fault-Tolerant Execution Framework for Transaction Processing

May 30, 2011
Toshio Suganuma, Akira Koseki, Kazuaki Ishizaki, Yohei Ueda, Ken Mizuno, Daniel Silva*, Hideaki Komatsu, Toshio Nakatani

IBM Research – Tokyo, *Amazon.com, Inc
Motivation

- Transaction-volume explosions are increasingly common in many commercial businesses
  - Online shopping, online auction services
  - Algorithmic trading
  - Banking services
  - more...

- It is difficult (if not impossible) to create systems that satisfy transaction, scalable performance, and high availability

- Can we improve performance without significant loss of availability?
Contributions

Study of performance-availability trade-off in a distributed cluster environment by proposing a new replication protocol

- Our replication protocol
  - Has a feature of continuous adjustment between performance and availability
  - Keeps global data consistency at transaction boundaries
  - Enables scalable performance with a slight compromise of availability
Motivation and Contributions

Replication Scheme
- Data replication model
- Existing replication strategies
- Our approach
- Replication protocol detail
- Failure recovery process
- Failover example

Availability

Experiment

Summary
Data tables are partitioned and distributed over a cluster of nodes.

Each partition is replicated on 3 different nodes (as Primary, Secondary, and Tertiary data), and each node serves for 3 different partitions.
Replication Scheme – Existing Approach

1. Synchronous replication
   – Primary waits for changes to be mirrored in Backup nodes
   – Allows failover without data loss
   – Example: Traditional RDB systems, e.g. DB2 parallel edition

2. Asynchronous replication
   – Primary proceeds without waiting acknowledgement from Backup
   – Risk data loss upon failover to Backup nodes
   – Better performance by passing synchronization delay to read transaction
   – Example: Chain replication [OSDI, 2004], Ganymed [Middleware, 2004]

A Chain example
We employ different replication policy for 2 backup nodes

- Primary: Active computation node
- Secondary: Synchronous replication node
- Tertiary: Asynchronous replication node

This allows performance improvement with relaxed synchronization, while Tertiary can contribute for increasing availability.
Protocol Detail – 1. Execute a Task

- Master sends messages to all Primary to start their local tasks
- Primary accumulates all data updates from application to logs and sends them to Secondary
- Secondary passes the change logs to Tertiary
# Protocol Detail – 2. Commit in Primary

- Secondary notifies Primary when log buffering is completed
- Primary commits the local transaction when log buffering completion message arrived from Secondary
- Primary then sends the task end message to Master
Protocol Detail – 3. Commit in Secondary

- Master receives task end messages from all Primary
- Master sends all Secondary to commit
- Primary start the next local task after receiving the message from Master

- Tertiary notifies Primary when the change logs are committed
- Primary deletes the corresponding logs
Node Failover Example

- A spare node is activated upon failure of any single node
  - Secondary and Tertiary are promoted to Primary and Secondary
  - Spare node gets copies from the new Secondary, and acts as Tertiary
Improved Availability by Tertiary

- Suppose both Primary and Secondary fail at the same time
- If Tertiary has the log records made in the last committed transaction, the system can continue without data loss
Data Loss Case

- If both Primary and Secondary fail at the same time, and
- If Tertiary has not received all the logs of the last committed transaction, some data is lost and the system is not automatically recoverable.

Diagram:

- Primary:
  - Update 1
  - Update 2
  - Update 3
  - Commit
  - Update 4

- Secondary:
  - Update 1
  - Update 2
  - Update 3
  - Commit

- Tertiary:
  - Delay
  - Update 1
  - Update 2
  - Incomplete logs
  - Not automatically recoverable

Current transaction:

- Commit
- Update 4

Last committed transaction:

- Update 1
- Update 2
- Update 3
Availability with Our Replication Scheme

- Availability of our system is affected by the delay of transferring the log to Tertiary.
- The delay is significantly affected by data transfer efficiency from Secondary to Tertiary
  - Disk accesses due to insufficient memory can be a bottleneck
- By removing I/O bottlenecks on the nodes, we can minimize the delay and maximize $P$, the probability of availability of the log records of the last committed transaction.

<table>
<thead>
<tr>
<th>1-synch-backup</th>
<th>P=0.5</th>
<th>2-synch-backup</th>
</tr>
</thead>
<tbody>
<tr>
<td>99.9%</td>
<td>99.99%</td>
<td>99.999%</td>
</tr>
<tr>
<td>99.99%</td>
<td>99.999%</td>
<td>99.9999%</td>
</tr>
</tbody>
</table>

**Case:** A cluster of 1,000 nodes, each has 0.001 failure probability (corresponding 3-year (= 1,000-day) MTBF and 1-day MTTR
Evaluation with TPC-C Workloads

- We created a batch job by combining three different scenarios in TPC-C; NewOrder, Payment, and Delivery

- We evaluate our replication protocol from the following aspects:
  - Scaling efficiency (strong scaling and weak scaling)
  - Replication overhead (with and without replication)
  - Effect of relaxed synchronization

Input: Purchase Order List

IBM BladeCenter JS21 (PowerPC970MP) x39
Strong Scaling Efficiency

- The throughput is increased almost linearly as nodes are added.
Weak Scaling Efficiency

- The execution time is almost flat as nodes are added if sufficient memory is available for the node (e.g. buffer pools of DB).
  - Otherwise, the increase of disk accesses causes the delay of synchronization.
Replication Overhead

- The replication overhead varies with the input data size per blade
  - A → heavy disk accesses causes fairly high overhead
  - B → with sufficient memory resource, the overhead is 20%

(a) Strong Scaling (# of record = 40,000)
Data Synchronization Effect

- We compared the total execution time of TPC-C NewOrder transactions between conventional (full synchronization) model and our relaxed synchronization model.

- The 43% reduction of the execution time is due to our approach of low synchronization overhead.

![Bar Chart]

- Conventional full synchronization
- Our relaxed synchronization
We proposed a new replication protocol that combines two different replication policies
  – Synchronous replication for Secondary and asynchronous replication for Tertiary.

Using our replication scheme:
  – We can achieve scalable performance
  – System tolerates up to 2 simultaneous node failure among triple redundant nodes most of the time
  – Overhead of data replication is 20% with sufficient memory

We showed performance-availability trade-off that we can obtain performance improvements by slightly compromising availability
  – E.g. 99.9999% → 99.999%
Backup
System Overview [Ishizaki et al, SYSTOR 2010]

1. Data (both DB tables and input files) are partitioned and distributed over a cluster of nodes, as specified by users.

2. Master partitions the job into tasks based on data layout, and assigns them to nodes based on owner-compute-rule.

3. Each node executes a task (which only requires local data accesses).
Replication Scheme in KVS

- Optimistic replication
  - Rely on eventual consistency model
  - Conflict resolution mechanism is necessary
  - Transaction cannot be supported (e.g., read-modify-write is not possible)
  - Superior in performance and availability
  - Example: Gossip protocol in Cassandra
Availability Calculation

Example

- A cluster of 1,000 nodes, each has the probability of 0.001 failure
  - E.g. 3-year (= 1,000-day) MTBF (Mean Time Between Failure) and 1-day MTTR (Mean Time To Repair)

- Conventional full synchronization approach:
  - System becomes unavailable only when all nodes holding a copy of a particular partition fail at the same time
  - 99.9999% availability for 2-backup-node replication
  - 99.9% availability for 1-backup-node replication

- Our relaxed synchronization approach:
  - System availability depends on the probability of log availability in tertiary on a simultaneous failure of primary and secondary nodes
  - 99.999% if we assume this probability of log-availability is 0.9
  - 99.99% if we assume this probability of log-availability is 0.5
What is “Sustainable State”?

- Primary has committed all updates in the last UOW and sent their logs to Secondary.
- Secondary has received the logs for the last UOW.
- Tertiary is alive and ready to receive logs

Our protocol proceeds by keeping the Sustainable State among all triplets

<table>
<thead>
<tr>
<th>Example</th>
<th>Transaction Boundary</th>
<th>Transaction Boundary</th>
<th>Transaction Boundary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UOW 1</td>
<td>UOW 2</td>
<td>UOW 3 (current)</td>
</tr>
<tr>
<td>Primary</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>All updates committed</td>
<td>All updates committed</td>
<td>All updates committed</td>
</tr>
<tr>
<td>Secondary</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>All change logs received</td>
<td>All change logs received</td>
<td>All change logs received</td>
</tr>
<tr>
<td>Tertiary</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>All change logs received</td>
<td>Receiving change logs</td>
<td></td>
</tr>
</tbody>
</table>
Failure Recovery Process

1. Find a transaction recovery point and determine new Primary and Secondary
2. Select a node to join the triplet as new Tertiary
3. Have the new Secondary send a snapshot and logs to the new Tertiary
4. Resume application on new Primary