Replication in distributed systems

Distributed Software Systems

Replication

- Motivation
  - Performance Enhancement
  - Enhanced availability
  - Fault tolerance
- Requirements
  - Replication transparency
  - Consistency
    - Depends upon application
    - In many applications, we want that different clients making (read/write) requests to different replicas of the same logical data item should not obtain different results
**System model**

- Assume replica manager apply operations to its replicas recoverably
- Set of replica managers may be static or dynamic
- Requests are reads or writes (updates)

**A basic architectural model for the management of replicated data**
System model

Five phases in performing a request

- Front end issues the request
  - Either sent to a single replica or multicast to all replica mgrs.
- Coordination
  - Replica managers coordinate in preparation for the execution of the request, i.e. agree if request is to be performed and the ordering of the request relative to others
    - FIFO ordering, Causal ordering, Total ordering
- Execution
  - Perhaps tentative
- Agreement
  - Reach consensus on effect of the request, e.g. agree to commit or abort in a transactional system
- Response

Total, FIFO and causal ordering of multicast messages

Notice the consistent ordering of totally ordered messages $T_1$ and $T_2$, the FIFO-related messages $F_1$ and $F_2$ and the causally related messages $C_1$ and $C_3$ — and the otherwise arbitrary delivery ordering of messages.
Group communication

- Role of group membership service
  - Provide an interface for group membership changes
  - Implement a failure detector
  - Notify members of group membership changes
  - Perform group address expansion

Services provided for process groups
**View delivery**

- A view reflects current membership of group
- A view is delivered when a membership change occurs and the application is notified of the change
  - Receiving a view is different from delivering a view
    - All members have to agree to the delivery of a view
- View-synchronous group communication
  - the delivery of a new view draws a conceptual line across the system and every message is either delivered on side or the other of that line

**View-synchronous group communication**

<table>
<thead>
<tr>
<th>Case</th>
<th>Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>a (allowed).</td>
<td><img src="#" alt="Diagram a" /></td>
</tr>
<tr>
<td>b (allowed).</td>
<td><img src="#" alt="Diagram b" /></td>
</tr>
<tr>
<td>c (disallowed).</td>
<td><img src="#" alt="Diagram c" /></td>
</tr>
<tr>
<td>d (disallowed).</td>
<td><img src="#" alt="Diagram d" /></td>
</tr>
</tbody>
</table>
Correctness Criterion for Replicated Objects

- Linearizability
  - The interleaved sequence of operations meets the specification of a single correct copy of the objects
  - The order of operations in the interleaving is consistent with the real times at which the operations occurred in the actual execution

- Sequential consistency replaces second criterion
  - The order of operations in the interleaving is consistent with the program order in which each individual client executed them

Example

Client 1

\[ X_1 = X_1 + 1; \]
\[ Y_1 = Y_1 + 1; \]

Client 2

\[ A = X_2; \]
\[ B = Y_2; \]

If \( A > B \)

print(A)

else.....
**Linearizable**

Client 1

\[
X = X + 1; \\
Y = Y + 1;
\]

Client 2

\[
A = X; \\
B = Y; \\
\text{If } (A > B) \\
\text{print}(A) \\
\text{else} 
\]

**Not linearizable but sequentially consistent**

Client 1

\[
X = X + 1; \\
Y = Y + 1; 
\]

Client 2

\[
A = X; \\
B = Y; \\
\text{If } (A > B) \\
\text{print}(A) \\
\text{else} 
\]
Neither linearizable nor sequentially consistent

Client 1

X = X + 1;
Y = Y + 1;

Client 2

A = X;
B = Y;

If (A > B)
print(A)
else

Fault-tolerant applications

- Passive (primary-backup) replication
  - Front ends only communicate with primary
  - Implements linearizability if primary is correct
  - If primary fails, it is replaced with a unique backup; replica managers that survive have to agree upon which operations had been performed when the replacement primary takes over
    - Requirements met if replica managers organized as a group and if primary uses view-synchronous communication to propagate updates to backups
The passive (primary-backup) model for fault tolerance

Fault tolerant applications cont’d

- Active replication
  - Front end multicasts request to each replica using a totally ordered reliable multicast
  - System achieves sequential consistency but not linearizability
    - Total order in which replica managers process requests may not be same as real-time order in which clients made requests
Active replication

Highly available services

- Emphasis on giving clients access to the service with reasonable response times, even if some results do not conform to sequential consistency
- Examples
  - Gossip
    - Relaxed consistency
      - Causal update ordering
  - Bayou
    - Pairwise anti-entropy protocol
    - Domain-specific conflict detection and resolution
  - Coda (file system)
    - Disconnected operation
Display from bulletin board program

<table>
<thead>
<tr>
<th>Item</th>
<th>From</th>
<th>Subject</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>A.Hanlon</td>
<td>Mach</td>
</tr>
<tr>
<td>24</td>
<td>G.Joseph</td>
<td>Microkernels</td>
</tr>
<tr>
<td>25</td>
<td>A.Hanlon</td>
<td>Re: Microkernels</td>
</tr>
<tr>
<td>26</td>
<td>T.L’Heureux</td>
<td>RPC performance</td>
</tr>
<tr>
<td>27</td>
<td>M.Walker</td>
<td>Re: Mach</td>
</tr>
</tbody>
</table>

Query and update operations in a gossip service
Front ends propagate their timestamps whenever clients communicate directly

A gossip replica manager, showing its main state components
Committed and tentative updates in Bayou

Tentative update $t_i$ becomes the next committed update and is inserted after the last committed update $c_N$.

Transactions on replicated data
One copy serializability

- Replicated transactional service
  - Each replica manager provides concurrency control and recovery of its own data items in the same way as it would for non-replicated data
  - Effects of transactions performed by various clients on replicated data items are the same as if they had been performed one at a time on a single data item
- Additional complications: failures, network partitions
  - Failures should be serialized wrt transactions, i.e. any failure observed by a transaction must appear to have happened before a transaction started

Replication Schemes

- Primary Copy
- Read one – Write All
  - Cannot handle network partitions
- Schemes that can handle network partitions
  - Available copies with validation
  - Quorum consensus
  - Virtual Partition
Replication Schemes cont’d

- Read-one write-all
  - Each write operation sets a write lock at each replica manager
  - Each read sets a read lock at one replica manager
- Two phase commit
  - Two-level nested transaction
    - Coordinator -> Workers
    - If either coordinator or worker is a replica manager, it has to communicate with replica managers
- Primary copy replication
  - ALL client requests are directed to a single primary server

Available copies replication

- Can handle some replica managers are unavailable because they have failed or communication failure
- Reads can be performed by any available replica manager but writes must be performed by all available replica managers
- Normal case is like read one/write all
  - As long as the set of available replica managers does not change during a transaction
Available copies replication

- Failure case
  - One copy serializability requires that failures and recovery be serialized wrt transactions
  - This is not achieved when different transactions make conflicting failure observations
  - Example shows local concurrency control not enough
  - Additional concurrency control procedure (called local validation) has to be performed to ensure correctness

- Available copies with local validation assumes no network partition - i.e. functioning replica managers can communicate with one another
Local validation - example

- Assume X fails just after T has performed GetBalance
  and N fails just after U has performed GetBalance
- Assume X and N fail before T & U have performed their Deposit operations
  - T’s Deposit will be performed at M & P while U’s Deposit will be performed at Y
  - Concurrency control on A at X does not prevent U from updating A at Y; similarly concurrency control on B at N does not prevent Y from updating B at M & P
  - Local concurrency control not enough!

Local validation cont’d

- T has read from an item at X, so X’s failure must be after T.
- T observes the failure of N, so N’s failure must be before T
  - N fails -> T reads A at X; T writes B at M & P
    - T commits -> X fails
  - Similarly, we can argue:
    - X fails -> U reads B at N; U writes A at Y -> U commits -> N fails
Local validation cont’d

- Local validation ensures such incompatible sequences cannot both occur
- Before a transaction commits it checks for failures (and recoveries) of replica managers of data items it has accessed
- In example, if T validates before U, T would check that N is still unavailable and X, M, P are available. If so, it can commit
- U’s validation would fail because N has already failed.

Network partition
Handling Network Partitions

- Network partitions separate replica managers into two or more subgroups, in such a way that the members of a subgroup can communicate with one another but members of different subgroups cannot communicate.

- Optimistic approaches
  - Available copies with validation

- Pessimistic approaches
  - Quorum consensus

Available Copies With Validation

- Available copies algorithm applied within each partition
  - Maintains availability for Read operations

- When partition is repaired, possibly conflicting transactions in separate partitions are validated
  - The effects of a committed transaction that is now aborted on validation will have to be undone
    - Only feasible for applications where such compensating actions can be taken
Available copies with validation cont’d

- Validation
  - Version vectors (Write-Write conflicts)
  - Precedence graphs (each partition maintains a log of data items affected by the Read and Write operations of transactions)
  - Log used to construct precedence graph whose nodes are transactions and whose edges represent conflicts between Read and Write operations
    - No cycles in graph corresponding to each partition
  - If there are cycles in graph, validation fails

Quorum consensus

- A quorum is a subgroup of replica managers whose size gives it the right to carry out operations
- Majority voting one instance of a quorum consensus scheme
  - R + W > total number of votes in group
  - W > half the total votes
  - Ensures that each read quorum intersects a write quorum, and two write quora will intersect
- Each replica has a version number that is used to detect if the replica is up to date.
Gifford’s quorum consensus examples

<table>
<thead>
<tr>
<th>Example</th>
<th>Replica 1</th>
<th>Replica 2</th>
<th>Replica 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency (milliseconds)</td>
<td>75</td>
<td>75</td>
<td>75</td>
</tr>
<tr>
<td>Voltage configuration</td>
<td>65</td>
<td>100</td>
<td>750</td>
</tr>
<tr>
<td>Quorum sizes</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Derived performance of file suite:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read Latency</td>
<td>65</td>
<td>75</td>
<td>75</td>
</tr>
<tr>
<td>Blocking probability</td>
<td>0.01</td>
<td>0.0002</td>
<td>0.000001</td>
</tr>
<tr>
<td>Write Latency</td>
<td>75</td>
<td>100</td>
<td>750</td>
</tr>
<tr>
<td>Blocking probability</td>
<td>0.0101</td>
<td>0.03</td>
<td></td>
</tr>
</tbody>
</table>

Virtual Partitions scheme

- Combines available copies and quorum consensus
- Virtual partition = set of replica managers that have a read and write quorum
- If a virtual partition can be formed, available copies is used
  - Improves performance of Reads
- If a failure occurs, and virtual partition changes during a transaction, it is aborted
- Have to ensure virtual partitions do not overlap
Two network partitions

Virtual partition
Two overlapping virtual partitions

Creating a virtual partition

Phase 1:
- The initiator sends a `Join` request to each potential member. The argument of `Join` is a proposed logical timestamp for the new virtual partition.
- When a replica manager receives a `Join` request, it compares the proposed logical timestamp with that of its current virtual partition.
  - If the proposed logical timestamp is greater it agrees to join and replies `Yes`;
  - If it is less, it refuses to join and replies `No`.

Phase 2:
- If the initiator has received sufficient `Yes` replies to have read and write quora, it may complete the creation of the new virtual partition by sending a `Confirmation` message to the sites that agreed to join. The creation timestamp and list of actual members are sent as arguments.
- Replica managers receiving the `Confirmation` message join the new virtual partition and record its creation timestamp and list of actual members.