Distributed Systems
ID2201

replication
Johan Montelius
The problem

- The problem we have:
  - servers might be unavailable
- The solution:
  - keep duplicates at different servers
Building a fault-tolerant service

• When building a fault-tolerant service with replicated data and functionality the service:
  – should produce the same results as a non-replicated service
  – should respond despite node crashes
  – ... if the cost is not too high
A first try...

- two replicas
- replica acknowledge operation then copy to peer
- front-end uses one replica and switch if replica fails
Let's see...

front-end directs requests to replica B

**Client 1**

```
setBalance_B(x, 10);
- B fails -
setBalance_A(y, 20);
```

**Client 2**

```
getBalance_A(y); -> 20
getBalance_A(x); -> 0
```
A fault-tolerant service - not

- This does not give us a correct service, but why?
- What are the requirements of a correct service?

Client 1

\[
\begin{align*}
\text{setBalance}_B(x,10); \\
\text{setBalance}_A(y,20);
\end{align*}
\]

Client 2

\[
\begin{align*}
\text{getBalance}_A(y); \rightarrow 20 \\
\text{getBalance}_A(x); \rightarrow 0
\end{align*}
\]
Correct behavior

- When talking about correct behavior we look at the sequence of operations and their returned values.

\[
\begin{align*}
\text{setBalance}_B(x, 10); \\
\text{setBalance}_A(y, 20); \\
\text{getBalance}_A(y); \\rightarrow 20 \\
\text{getBalance}_A(x); \\rightarrow 0
\end{align*}
\]
What is a correct behavior

• A replicated service is said to be linearizable if for any execution there is some interleaving that ... 
  – meets the specification of a non-replicated service
  – matches the real time order of operations in the real execution
A less restricted

- A replicated service is said to be **sequentially consistent** if for any execution there is *some interleaving* that ...
  - meets the specification of a non-replicated service
  - matches the *program order* of operations in the real execution
Sequential consistency

- Can we find a interleaving with a correct behavior.

**Client 1**

```plaintext
setBalance_B(x, 10);
setBalance_A(y, 20);
```

**Client 2**

```plaintext
getBalance_A(y); -> 20
getBalance_A(x); -> 0
```
Sequentially consistent

- Is this behavior sequentially consistent?
- Linearizable?

**Client 1**

```
setBalance_B(x, 10);
setBalance_A(y, 20);
```

**Client 2**

```
getBalance_A(y); -> 0
getBalance_A(x); -> 0
```
System model

- Asynchronous system, nodes fail only by crashing, no network partitions.
Group membership service

Distributed Systems ID2201
Life of a request

- Request
  - front-end delivers a request
- Coordination
  - replica managers decide on order
- Execution
  - tentative execution that can be aborted
- Agreement
  - reaching a consensus
- Response
  - front end collect one or more responses
passive replication
passive replication

- **Request**
  - request with a unique identifier
- **Coordination**
  - checks if it is a new request
- **Execution**
  - Execute, and store response
- **Agreement**
  - send updated state and reply to all backup nodes
- **Respond**
Is it linearizable?

- The primary replica manager will serialize all operations.
- If the primary fails, it retains linearizability if backup takes over where the primary left off.
primary crash

• Primary crash:
  – backups will receive *new view* with primary missing
  – new primary is elected

• Request is resent:
  – if agreement was reached last time, the reply is known and is resent
  – if not, the execution is redone
The rôle of the front end

Linearizable with several front ends?
active replication

front end

state machines

RM

RM

RM
Active replication

- **Request**
  - multicast to group, unique identifier

- **Coordination**
  - deliver request in *total order*

- **Execution**
  - all replicas are identical

- **Agreement**
  - not needed

- **Response**
  - sent to front end, first reply to client
Active replication

• Sequential consistency:
  – All replicas execute the same sequence of operations.
  – All replicas produce the same answer.

• Linearizability:
  – Total order multicast does not (automatically) preserve real-time order.
High availability

- Both replication schemes require that servers are available.
- If a server crashes it will take some time to detect and remove the faulty node.
  - depends on network
  - is this acceptable
- Can we build a system that responds even if all nodes are not available?
Gossip architecture
Relaxed consistency

- Increase availability at the expense of consistency.
  - causal update ordering
  - forced (total and causal) update ordering
  - Immediate update ordering (total order with respect to all other updates)
Example: bulletin board

- Adding messages:
  - causal ordering
- Adding a user:
  - forced ordering
- Removing a user:
  - immediate ordering
- All replicas should agree on what messages are before the removal of the user.
Front ends are active

- Front ends keep a vector clock that reflects the last seen value.
- The vector holds an entry for each replica in the system.
- The vector clock is updated as the front end sees replies from the replicas.
- The front end is responsible for fault tolerant replication.
I have seen values written at <2,4,6>, don't give me old data.

Ok, this is the value at <2,5,6>
Query

- A front end sends a query request to any replica manager.
- Query contains vector time stamp.
- Replica manager must hold query until it has all information that *happen_before* the query.
- Replica manager returns response and new time stamp.
Quorum based

- Can we have a static group (nodes might fail but they will be restarted) and solve consistency using a quorum.
- Why would we like to do this?
Updates

I have seen values written at \(<3,5,7>\), write this later.

write X at \(<3,5,7>\)

ack: \(<4,5,7>\)

replica manager

replica manager

replica manager

front end

<3,5,6>

<2,5,7>

<3,5,6>
Updates

- Front end sends updates to one (or more) replica manager.
- The update is scheduled by the replica manager to be executed in causal order.
- Updates is sent to remaining replica managers using the gossip protocol.
Gossip architectures

- Performance at the price or causal consistency.
- Forced and immediate consistency more expensive.
- Can the application live with causal consistency?
- Highly available, only one replica needs to be available.
Quorum based

Each datum has a write time (logical or real) to determine the most recent
Sequential consistent

- Assume the read or write quorum must be taken in order for the operation to take place.
- \( R + W > N \)
  - A read operation will overlaps with the most recent write operation.
- \( W > N/2 \)
  - Two read operations can not occur concurrently.
Summary

- Replicating objects used to achieve fault tolerant services.
- Services should (?) provide single image view as defined by sequential consistency.
- Passive replication
- Active replication
- Quorum based replication