LTAT.06.007 Distributed Systems
Lecture 7 – Replication

Huber Flores, PhD
ASSOCIATE PROFESSOR

Tartu, Estonia, Delta building (auditorium 1021) 21/03/2022
Recap

- **Election algorithm**
  - Selecting a leader from a group (election steps)
  - Different algorithms

- **Leader coordination**
  - Mutual exclusion
Think about

Back-up nodes

Online book shop

Charge

Success

Server 1

Service payment

Leader

Server 2

Service payment

Replica

Server 3

Service payment

Replica

Number of replicas?
Agenda

• **Goal:** To study the importance of replication in distributed systems

• **Content:**
  - Reasons of replication
  - Replica caching, CDNs
  - Replication issues (Idempotence, timestamps and tombstones)
  - Quorum

**After this lecture, you should be able to:**

• To explain the role of replication in distributed systems

• To understand the trade-off that replication has on system performance
Reasons for replication
Reasons for replication
Example: web hosting systems

Adaptation triggering

• When and how adaptation are triggered?
  ▪ Often based on periodic evaluation of performance metrics

• Periodic evaluation may miss sudden changes such as flash crowds
  ▪ Sudden burst of requests for a specific web document that can bring an entire web site down

• Flash crowds are difficult to deal with
  ▪ Overprovisioning is expensive
  ▪ Flash-crowd predictor for giving enough time for dynamic replication before the flash crowd (Flash crowds can have very different access patterns)
Reasons for replication
Reasons for replication
Reasons for replication

Essence

• Copy of the same data in multiple nodes
  ▪ Databases (Apache Cassandra), file systems (NFS – Network file sharing)...
• A node that has a copy of the data is called **replica**
• If some replicas are faulty, others are still accessible
• Spread load across replicas
• If data does not change, data distribution becomes relatively easy

One source of storage is not enough to ensure reliability (replicate, replicate)

RAID "redundant array of inexpensive disks“ (Different levels – Level 0)

Compare to RAID

RAID has a single controller, in a distributed system, each node acts independently

[source]  
https://blocksandfiles.com/2019/07/12/the-reasons-disk-drives-fail/
## Reasons for replication

### Essence

- **Enhancing reliability**
  - Protection against hardware crashes and corruption of data

- **Improving performance**
  - Scaling in terms of numbers and geographical area
  - Caching is a special form of replication
  - Trade-off between performance and scalability

### Challenges (Keep replicas consistent)

- When one copy is updated, other copies need to be updated, as well
- A number of consistency models
- Protocols for distribution of updates
System performance and replication

Essence

Replication improves server performance, particularly reliability and availability.

However, if data is not replicated properly, this can lead to a system that cannot be utilized in practice.
Replica caching

Organizing information
Analogy: Library of books

“The memory hierarchy”

The idea of keeping around pieces of information that you utilize frequently
Replica caching

Publish and subscribe architecture, e.g., trigger on events
Replica caching

Cache replacement or (Cache eviction)

New update e.g., DNS?
Replica caching

Essence

Cache eviction policies

- First-in, First out (FIFO) – evict or overwrite whatever has been sitting in the cache the longest
- Random eviction – adding new data to the cache and overwriting old data at random
- Least recently used (LRU) – evicting the data item that has gone the longest untouched

- In the context of distributed systems, Content Distributed Networks (CDNs)
  - Computer around the world that contain copies of popular websites (in cache)

Cache order?

Self-organizing lists?
## Replica caching

### Essence

#### Cache eviction policies
- First-in, First out (FIFO) – evict or overwrite whatever has been sitting in the cache the longest
- Random eviction – adding new data to the cache and overwriting old data at random
- Least recently used (LRU) – evicting the data item that has gone the longest untouched

- In the context of distributed systems, Content Distributed Networks (CDNs)
  - Computer around the world that contain copies of popular websites (in cache)

### Cache order?

**Self-organizing lists:** Principle that suggests that chaos is one of the most well-designed and efficient structures available.
- Analogy: your pile of papers in your desk
Replica caching

Examples

<table>
<thead>
<tr>
<th>CDNs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CloudFare</td>
<td><a href="https://www.cloudflare.com/">https://www.cloudflare.com/</a></td>
</tr>
<tr>
<td>Incapsula</td>
<td><a href="https://www.incapsula.com">https://www.incapsula.com</a></td>
</tr>
<tr>
<td>StackPath</td>
<td><a href="https://www.stackpath.com/maxcdn">https://www.stackpath.com/maxcdn</a></td>
</tr>
<tr>
<td>Sucuri</td>
<td><a href="https://sucuri.net/website-performance/">https://sucuri.net/website-performance/</a></td>
</tr>
<tr>
<td>Imperva</td>
<td><a href="https://www.imperva.com/">https://www.imperva.com/</a></td>
</tr>
<tr>
<td>Many others...</td>
<td></td>
</tr>
</tbody>
</table>
Data changes (Consistency)

Essence

Tracking and managing changes on the data is a key problem in replication.

The system must identify which requests have been already seen.
Data changes (Consistency)

CWBChicago
@CWBChicago

-1 Following  -1 Followers

Profile
Idempotence

Essence

- **Idempotence**: an operation that has not additional effect in a final result even after applied multiple times.
  - Requests can be retried without deduplication
  - A function $f$ is idempotent if $f(x) = f(f(x))$
  - **Not idempotent**: $f(\text{likeCount}) = \text{likeCount} + 1$
  - **Idempotent**: $f(\text{likeSet}) = \text{likeSet} \cup \{\text{userID}\}$

Choice of retry behavior:

- **At-most-once** semantics: send request, don’t retry, update may not happen
- **At-least-once** semantics: retry request until acknowledged, many repeat update
- **Exactly-once** semantics: retry + idempotence or deduplication
Adding and then removing again

Essence

\[ f(\text{likes}) = \text{likes} \cup \{\text{userID}\}; \ g(\text{likes}) = \text{likes} \setminus \{\text{userID}\}; \text{idempotent?} \]
Adding and then removing again (2)

Client 1 → add(x) → Server A (data) → add(x) → remove(x) → Server B (data) → remove(x)
Adding and then removing again (3)

Client 1 → Server A (data) → Server B (data)

add(x)

add(x)
Timestamps and tombstones

“remove(x)” does not actually remove x: it labels x with “false” to indicate it is invisible (a tombstone)
## Replication issues

### Main issue
To keep replicas consistent, we generally need to ensure that all **conflicting** operations are done in the same order everywhere.

### Conflict operations
- **Read–write conflict**: a read operation and a write operation act concurrently
- **Write–write conflict**: two concurrent write operations

### Issue
Guaranteeing global ordering on conflicting operations may be a costly operation, downgrading scalability. **Solution**: weaken consistency requirements so that hopefully global synchronization can be avoided.
Reconciling replicas

**Essence**

Replicas periodically communicate among themselves to check for any inconsistencies. (Anti-entropy protocol)

\[
\{x \rightarrow (t_2, \text{false})\} \quad \{x \rightarrow (t_1, \text{true})\}
\]

\[
t_1 < t_2
\]

\[
\{x \rightarrow (t_2, \text{false})\} \quad \{x \rightarrow (t_2, \text{false})\}
\]

Update the record (data item) with latest timestamps
Concurrent writes by different clients

Two common approaches
- Last writer wins (LWW)
  - Use timestamps with total order (e.g., Lamport clock)
- Multi-value register
  - Use timestamps with partial order (e.g., Vector clock)
Replica failure

Observation
A replica may be unavailable due to network partition or node fault, etc.

Assume

- Each replica has a probability $p$ of being unavailable (faulty)
- Probability of all $n$ replicas being faulty: $p^n$
- Probability of $\geq 1$ out of $n$ replicas being faulty: $1 - (1 - p)^n$

Example with $p=0.01$

<table>
<thead>
<tr>
<th>Replicas $n$</th>
<th>$P(\geq1$ faulty$)$</th>
<th>$P(\geq(n+1)/2$ faulty$)$</th>
<th>$P(\text{all } n$ faulty$)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>3</td>
<td>0.03</td>
<td>$3 \times 10^{-4}$</td>
<td>$10^{-6}$</td>
</tr>
<tr>
<td>5</td>
<td>0.049</td>
<td>$1 \times 10^{-5}$</td>
<td>$10^{-10}$</td>
</tr>
<tr>
<td>100</td>
<td>0.63</td>
<td>$6 \times 10^{-74}$</td>
<td>$10^{-200}$</td>
</tr>
</tbody>
</table>
Replica failure

Observation
A replica may be unavailable due to network partition or node fault, etc.

Assume
- Each replica has a probability $p$ of being unavailable (faulty)
- Probability of all $n$ replicas being faulty: $p^n$
- Probability of $>= 1$ out of $n$ replicas being faulty: $1 - (1 - p)^n$

Example with $p=0.01$

<table>
<thead>
<tr>
<th>Replicas $n$</th>
<th>$P (&gt;=1$ faulty$)$</th>
<th>$P (&gt;= (n+1)/2$ faulty$)$</th>
<th>$P (all n$ faulty$)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>3</td>
<td>0.03</td>
<td>$3 \times 10^{-4}$</td>
<td>$10^{-6}$</td>
</tr>
<tr>
<td>5</td>
<td>0.049</td>
<td>$1 \times 10^{-5}$</td>
<td>$10^{-10}$</td>
</tr>
<tr>
<td>100</td>
<td>0.63</td>
<td>$6 \times 10^{-74}$</td>
<td>$10^{-200}$</td>
</tr>
</tbody>
</table>
Read-after-write consistency

Client

Write x
\( t_1, \text{set } x = v_1 \)

A

B
Read-after-write consistency

Client $t_1$ $(t_1, \text{set}(x = v_1))$
Read-after-write consistency

Client

A

B

t_1

Read x

get(x)

(t_0, v_0)
Replicated-write protocols

<table>
<thead>
<tr>
<th>Essence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write operations can be carried out at multiple replicas</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Active replication</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Each replica has an associated process carrying out update operations</td>
</tr>
<tr>
<td>• Drawback: operations need to be carried out in same order everywhere</td>
</tr>
<tr>
<td>▪ Lamport timestamps (does not scale well in large systems)</td>
</tr>
<tr>
<td>▪ Central coordinator (sequencer)</td>
</tr>
<tr>
<td>• Problem with central coordinator: replicated invocations</td>
</tr>
<tr>
<td>▪ Coordinator in each replica for managing invocations and replies</td>
</tr>
</tbody>
</table>
Replicated-write protocols

Quorum-based protocols

Majority (among the replicas) based mechanisms

Client

A

B

C
Replicated-write protocols

Quorum-based protocols

2 out of 3 quorum

Write x
\(t_1, \text{set}(x = v_1)\)

Client

A

B

C

\(t_1, \text{set}(x = v_1)\)
Replicated-write protocols

Quorum-based protocols

2 out of 3 quorum

Client A B C

Value set

ok

ok

ok

37
Replicated-write protocols

Quorum-based protocols

2 out of 3 quorum

Client looks at the timestamp to figure out latest value
Replicated-write protocols

Quorum-based protocols

• Clients request and acquire permission of multiple servers before either reading or writing replicated data item

• Gifford’s scheme for N replicas
  ▪ Read quorum: client needs permission from arbitrary $N_R$ servers
    Prevents read-write conflicts
  ▪ Write quorum: client needs permission from arbitrary $N_W$ servers
    Prevents write-write conflicts
  ▪ Constraints:
    ▪ $N_R + N_W > N$
    ▪ $N_W > N/2$
Replicated-write protocols

Quorum-based protocols

Read quorum and write quorum share >= 1 replica
Replicated-write protocols

Essence

(a) Correct choice of read and write set.
(b) May lead to write-write conflicts
(c) Correct choice known as ROWA (read one, write all)
Read repair

Client-support

Update \((t_1, v_1)\) is more recent than \((t_0, v_0)\) since \(t_0 < t_1\)

Client helps propagate \((t_1, v_1)\) to other replicas
Summary

• Studied the importance of replication for improving system performance
• Explored the challenges associated with replication and reconciling replicas
• Studied Quorums (Read and Write)
References

Part of this material is inspired by:

• Distributed Systems course given by Dr. Martin Kleppmann (University of Cambridge, UK)
• Understanding Distributed Systems, Version 1.1.1., Roberto Vitillo, 2021
• Algorithms to live by, Brian Christian and Tom Griffiths
Next lecture

Consensus
Questions?

E-mail: huber.flores@ut.ee