LTAT.06.007 Distributed Systems
Lecture 10 – Consistency and Replication II

Huber Flores, PhD
ASSOCIATE PROFESSOR

Tartu, Estonia 19/04/2021
Recap

• Learned the relation between consistency and replication
• Studied different consistency models
Roadmap (so far)

Logical order of events

Global clock

Internal clock in components

1. a
2. b
3. P_k
4. Amazon.com
5. DNS
6. ut.ee
7. printer1
8. Internal clock in components
Agenda

• **Goal:** To learn the management of replicas in distributed systems

• **Content:**
  - Replica management
    - Content replication and placement
    - Content distribution
  - Consistency protocols
    - Continuous consistency and primary-based protocols

After this lecture, you should be able to:

• Understand the distribution of content in different replicas.
Replica management
Replica management

Essence

• Where, when and by whom replicas should be placed?
  ▪ Placing of replica servers
  ▪ Placing of content

• Which mechanisms to use for keeping the replicas consistent?
  ▪ Consistency protocols
Replica management

Leisure hours

Working hours
Replica management
Replica management
Replica placement

Essence (Figure out what the best K places are out of N possible locations)

• Select best location out of N - K for which the average distance to clients is minimal. Then choose the next best server. (Note: The first chosen location minimizes the average distance to all clients.) Computationally expensive.

• Select the K-th largest autonomous system and place a server at the best-connected host. Computationally expensive.

• Position nodes in a d-dimensional geometric space, where distance reflects latency. Identify the K regions with highest density and place a server in every one. Computationally cheap.
Replica placement

Replica-server placement

Too small

Too large

Cell

Just right
# Content replication and placement

## Distinguish different types

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Permanent replicas</strong></td>
<td>- Few replicas, statically configured</td>
</tr>
<tr>
<td></td>
<td>- Replication across a limited number of servers on a single LAN</td>
</tr>
<tr>
<td></td>
<td>▪ Mirroring using mirror sites geographically spread across Internet</td>
</tr>
<tr>
<td><strong>Server-initiated replicas</strong></td>
<td>- Server keeps track of client requests</td>
</tr>
<tr>
<td></td>
<td>▪ Dynamic migration/replication of files to servers nearby active clients</td>
</tr>
<tr>
<td><strong>Client-initiated replicas</strong></td>
<td>- Client stores local copies of data to improve access times</td>
</tr>
<tr>
<td></td>
<td>▪ Client caches can be shared (e.g. institutional web proxy)</td>
</tr>
</tbody>
</table>
Replica placement
Content replication and placement

The logical organization of different kinds of copies of a data store into three concentric rings
Dynamic Web content placement

Counting access requests from different clients

- Keep track of access counts per file, aggregated by considering server closest to requesting clients
- Number of accesses drops below threshold $D \rightarrow$ drop file
- Number of accesses exceeds threshold $R \rightarrow$ replicate file
- Number of access between $D$ and $R \rightarrow$ migrate file
Content distribution

Consider only a client-server combination

- Propagate only notification/invalidation of update (often used for caches)
- Transfer data from one copy to another (distributed databases): passive replication
- Propagate the update operation to other copies: active replication

Note

No single approach is the best, but depends highly on available bandwidth and read-to-write ratio at replicas.
## Content distribution

### Pull versus push updates

<table>
<thead>
<tr>
<th>Type of Protocol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Push-based (server-based) protocols</strong></td>
<td>Updated are propagated to other replicas even without their request. Often used between permanent and server-initiated replicas. When high degree consistency (identical replicas) is required. Efficient when read-to-update ratio is high.</td>
</tr>
<tr>
<td><strong>Pull-based (client-based) protocols</strong></td>
<td>Often used by client caches, polls server to see if update is needed. Efficient when read-to-update ratio is relatively low.</td>
</tr>
<tr>
<td><strong>Hybrid form based on lease</strong></td>
<td>Server pushes updates to client for a specified time.</td>
</tr>
</tbody>
</table>
Content distribution

Push-based

Pull-based
Comparison between push-based and pull-based protocols in case of multiple clients, single server systems

- Pushing updates: server-initiated approach, in which update is propagated regardless whether target asked for it.
- Pulling updates: client-initiated approach, in which client requests to be updated.

<table>
<thead>
<tr>
<th>Issue</th>
<th>Push-based</th>
<th>Pull-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>State at server</td>
<td>List of client replicas and caches</td>
<td>None</td>
</tr>
<tr>
<td>Messages sent</td>
<td>Update (and possibly fetch update later)</td>
<td>Poll and update</td>
</tr>
<tr>
<td>Response time at client</td>
<td>Immediate (or fetch-update time)</td>
<td>Fetch-update time</td>
</tr>
</tbody>
</table>
**Content distribution: client/server system**

**Observation**
We can dynamically switch between pulling and pushing using leases: A contract in which the server promises to push updates to the client until the lease expires.

**Make lease expiration time dependent on system’s behavior (adaptive leases)**

- **Age-based leases**: An object that hasn’t changed for a long time, will not change in the near future, so provide a long-lasting lease.

- **Renewal-frequency based leases**: The more often a client requests a specific object, the longer the expiration time for that client (for that object) will be.

- **State-based leases**: The more loaded a server is, the shorter the expiration times become.
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 ($\geq 1$ faulty)</th>
<th>P ($\geq (n+1)/2$ 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>
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>$P$ (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-write consistency

Write $x$ at $T_0$, set $x = v1$
Read-write consistency

Write x
$T_0. \text{set } x = v1$
Read-write consistency

Write $x$
$T_0$, set $x = v1$

Read $x$
$T_0$, $V0$
Consistency protocols
Consistency protocols

**Definition**

Consistency protocol describes an implementation of a specific consistency model.
Continuous consistency

Data-centric protocols

- Three continuous consistency ranges: numerical deviations, staleness deviations, ordering deviations

- Bounding numerical deviations (Yu and Vahdat 2000)
  - Write operations are assigned weights
  - Each server keeps log of writes it has performed on its own local copy
  - Writes are propagated to other servers, e.g. using an overlay network protocols
  - Weights and write operations contribute to a view (state) of values at a particular server
  - A server K may advance the view of server L

- Bounding staleness deviations with vector clocks

- Bounding ordering deviations by enforcing globally consistent ordering of tentative writes with primary based or quorum-based protocols
Primary-based protocols

Essence

• Each data item x in data store has an associated primary which is responsible for coordinating write operations on x

• Remote-write: primary is fixed at a remote server

• Local-write: primary is moved to the process where write operation is initiated and write is carried out locally

• Provide straightforward implementation of sequential consistency
  ▪ Primary can order all incoming writes in a globally unique time order
## Remote-write primary-based protocols

### Characteristics

- Read operations performed on local copy
- Write operations forwarded to a (fixed) primary copy
- Updates implemented as a blocking or nonblocking operation
  - If blocking, process initiating update may be blocked for long time
- Primary can order all writes in globally unique order
  - If blocking updates, processes will always see the effects of their most recent write operation
Remote-write primary-based protocols

Primary-backup protocol

- W1. Write request
- W2. Forward request to primary
- W3. Tell backups to update
- W4. Acknowledge update
- W5. Acknowledge write completed
- R1. Read request
- R2. Response to read
Local-write primary-based protocols

Primary copy migrates between processes performing write operations

• Successive writes can be carried out locally
• Reading processes can still access their local copy
• Nonblocking protocol needed for propagating updates to replicas after primary has locally performed the updates
Local-write primary-based protocols

Primary-backup protocol with local writes

- W1. Write request
- W2. Move item x to new primary
- W3. Acknowledge write completed
- W4. Tell backups to update
- W5. Acknowledge update

- R1. Read request
- R2. Response to read
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 (<strong>sequencer</strong>)</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
Replicated-write protocols

Quorum-based protocols

2 out of 3 quorum

Client

Write x
T₀, set x = v₁

A

B

C
Replicated-write protocols

Quorum-based protocols
2 out of 3 quorum

Client

T₀, set x = v₁

Value set

A

B

C

ok

ok
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

Read quorum: A, B, C
Write quorum: C, D
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)
Summary

What we learned?

• Techniques for replication
• The relation between replication and consistency
• Replication as a way to provide more reliability and scalability for distributed systems
Next lecture

Fault tolerance I
Questions?

E-mail: huber.flores@ut.ee