Frequently asked questions from the previous class survey
Topics covered in this lecture

- Replica Management
- Consistency protocols
- Primary based protocols
- Replicated write protocols
Key issues in replication within a distributed system

- **Placement of replicas**
  - When, where and by whom
- **Placement of content**
- **Mechanisms** to keep replicas consistent

Replica-Server Placement

- $K$ out of $N$ possible locations ($K < N$)
- Optimization problem
  - Computationally complex
- Solved only through **heuristics**
Replica-Server placements:
Distance between client and locations

- In terms of latency or bandwidth
- If $k$ servers have been placed, there are $N-k$ locations
  - Select server one at a time
  - Minimize average distance between server and its client

Replica Server placements
Use Internet topology

- **Autonomous Systems**
  - Network in which all nodes run the same protocol
  - About 20,000 AS
- Within an AS place server on router with largest NIC/links
  - Recursively repeat for other AS
Problems with these approaches

- Computing placements is expensive
- Does not work well with flash-crowds

Arriving at quick replica placements

- **Region/cell** is a collection of nodes accessing the same content
  - Inter-node latency is low

- Select most demanding regions
  - With largest number of nodes
  - One of the nodes acts as a replica
Quick replica server placements

- Nodes are in an $m$-dimensional geometric space
- Identify $K$ largest clusters
  - Assign node from each cluster to host replica

Types of Replicas
Types of Replicas

- Permanent Replicas
  - Initial set of replicas that comprise data store
    - Usually a small set
  - Files stored across servers at a single location
    - Request forwarded using round-robin strategy
  - Files copied to mirror sites
    - Geographically dispersed
Server initiated replicas

- Copies that exist to *enhance* performance
- Created at the *initiative* of the owner of data store

Server initiated replicas: Example

- Web server in NYC
  - Can handle dissemination loads effectively
- **Bursts** of traffic over 2-3 days may come in
  - From some specific location (or set of locations)
- Install *temporary replicas* in regions where requests originate
Server initiated replicas:
Issues in dynamic replications

- Replication takes place to **reduce load** at server
- **Specific** files on server migrated/replicated to servers in **proximity** of requesting clients

Dynamic replication: Migrating/replicating files

- Each server tracks **access counts** per file
  - And also who initiates accesses
- Given a client C
  - Each server can determine which of the servers is closest to C
Counting access requests from clients: C1 and C2 share closest server P

- Accesses from C1, C2 for file F at server Q are registered as if they are from P
  - $\text{count}_Q(P, F)$

Replication threshold: $\text{rep}(S, F)$

- Indicates number of requests for file is high
- Might be worth replicating it
Deletion thresholds

- When requests for file $F$ at server $S$ drops below deletion threshold, $\text{del}(S,F)$
  - File $F$ removed from $S$
- Number of replicas reduce
- Higher loads at the other servers
- Ensure at least one copy of file continues to exist

More on replication and deletion thresholds

- $\text{rep}(S, F)$ always chosen to be higher than the $\text{del}(S, F)$
- If a number of requests lie between deletion and replication threshold
  - File can only be migrated
  - Number of replicas for file should be the same
Reevaluating the placement of files at a server Q

- Check **access count** for each file
- If number of accesses < \( del(Q, F) \) ?
  - File deleted unless it is the last copy
- For some server \( P \), if \( count_Q(P, F) \) is more than \( \frac{1}{2} \) of requests for \( F \) at \( Q \)?
  - Server \( P \) is requested to **take over** copy of \( F \)
  - Migration

Migration/replication of a file may not always succeed

- Server \( P \) might already be heavily overloaded
- \( Q \) will then attempt to replicate \( F \) **elsewhere**
  - Number of access > \( rep(Q, F) \)
- If \( count_Q(R, F) \) exceeds a certain fraction of all requests for \( F \) at \( Q \)
  - Try to replicate at \( R \)
Client initiated replicas:
Client cache

- Temporarily store data that was just requested
  - Could be on client’s machine or nearby machine
- Used to improve access times
- Data kept in cache for a limited time
  - Avoid stale data problem
  - Make room for other data
- To improve cache hits; cache may be shared between clients

CONSISTENCY PROTOCOLS
Consistency Protocols

- Describes an implementation of a consistency model
- Models we looked at
  - Data centric
  - Client centric

Continuous consistency protocols:
Bounding

- Numerical deviations
- Staleness deviations
- Ordering deviations
Continuous consistency:
Bounding numerical deviations

- Writes on a single data item $x$
- Each write is submitted to one of $N$ available servers

Several submitted writes need to be propagated to all servers

- TW[i, j]: write executed by server $S_i$ that originated at $S_j$
- TW[i, i] aggregate writes submitted to $S_i$
  - By clients
Contrasting the values of $x$

- Actual value $v$ of $x$ at time $t$: $v(t)$
  - $v(0)$ is the initial value of $x$
- $v(t)$ is determined by all the submitted writes

$$v(t) = v(0) + \sum_{k=1}^{N} TW[k,k]$$

$$v_i = v(0) + \sum_{k=1}^{N} TW[i,k]$$

Bounding numerical deviations

- The upper bound on deviations that needs to be enforced
  $$v(t) - v_i \leq \delta_i$$
- Server $S_i$ propagates updates from $S_j$ to $S_k$
  - $S_k$ maintains a view $TW_k[i,j]$
- When $S_k$ notices $S_i$ has not been keeping up with updates
  - Propagate updates from its log
Bounding staleness deviation

- Maintain real-time vector clock
  - \( RVC_k[i] = T(i) \)
    - \( S_k \) has seen all writes at \( S_i \) up to time \( T_i \)
    - \( T_i \) is time \textit{local} to \( S_i \)
  - At \( S_k \) when \( T(k) - RVC_k[i] > \text{staleness\_limit} \)
    - \textbf{Pull} writes at \( S_i \) with timestamp \( > RVC_k[i] \)

Bounding ordering deviations

- Each server has a \textit{local copy} of \textit{tentative} writes
  - Actual order in which they should be applied is not yet determined
- Limit on the maximum length of the queue
When the maximum queue length limit is reached

- No new writes will be accepted
- Attempt to **commit pending writes**
  - Determine *order* of these writes from other servers

**PRIMARY BASED PROTOCOLS**
Primary based protocols

- Used in enforcing sequential consistency
- Each data item $x$ has an associated primary
  - Responsible for coordinating writes on $x$
- Primary may be
  - **Fixed** at a remote server
  - **Moved** to process where write operations are performed

Primary-based protocols:
Remote-Write \{Primary backup\}

- To perform write operation on $x$
- Forward write to primary-server for $x$
- Primary performs
  - Update on its local copy of $x$
  - Forwards update to the backup servers
Remote-write protocols:
Primary back-up protocol

Local-write protocols

- Primary copy migrates between process
  - From current server to the one wanting to update

- To perform write operation on \( x \)
  - Locate primary copy for \( x \)
  - Move primary copy to its own location
Local-write protocol:
Primary back-up protocol

Data Store

Old primary for item x

New primary for item x

R1

W2

W4

W4

W4

W1

W3

W4

W4

R2

Replicated Write Protocols
Replicated write protocols

- Write operations are carried out at multiple replicas
  - Not just 1 (or primary)
- Active Replication
  - Operation forwarded to all replicas
- Quorum-based
  - Based on majority voting

Active Replication

- Operation is sent to each replica
- Must be carried out in same order everywhere
  - Lamport’s clocks
  - Use of a central coordinator: Sequencer
    - Could start to resemble primary-based protocols
Quorum-based protocols:
Clients must request and acquire permissions

- From **multiple** servers
- **Before** reading and writing replicated data items

Quorum-based protocols:
Distributed File System example \{Write\}

- File is replicated on \( N \) servers
- To update a file
  - Client must contact at least \( \frac{N}{2} + 1 \) servers
  - Majority
  - Get them to agree to do the update
- Upon agreement
  - File is changed and version number incremented
Quorum-based protocols:
Reading a replicated file

- Client must contact at least \((N/2 + 1)\) servers
  - Ask them for version numbers of file
- If version numbers agree ... most recent version
- With \(N=5\), and
  - Clients see 3 responses with version-8
  - Then getting 2 responses with version-9?
    - Impossible, because update to version-9 needs 3 to agree

Quorum-based protocols:
When there are \(N\) replicas

- Read quorum \(N_R\)
- To modify a file write-quorum \(N_W\)
  - \(N_R + N_W > N\)
    - Prevent read-write conflict
- \(N_W > N/2\)
  - Prevent write-write conflict
Quorum-based protocols:
Example 1

- Read Quorum: [Blue]
- Write Quorum: [Red]

N_R = 3  N_W = 10

Write-write conflict
Concurrent writes to 
\{A, B, C, F, G\} and \{D, H, I, J, K, L\}
will be accepted

Quorum-based protocols:
Example 2

- Read Quorum: [Blue]
- Write Quorum: [Red]

N_R = 1  N_W = 12

November 5, 2019
Professor: SHRIDEEP PALICKARA
Dept. Of Computer Science, Colorado State University
The contents of this slide-set are based on the following references