15-440
Distributed Systems
Recitation 6

Tamim Jabban

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Project 2

- Involves *building on your Project 1* *Distributed File System* (DFS): *FileStack*
- P2_StarterCode: Copy files into your P1 folder
- Release Date: October 4th
- Due date: October 21st

Project 1: Recap

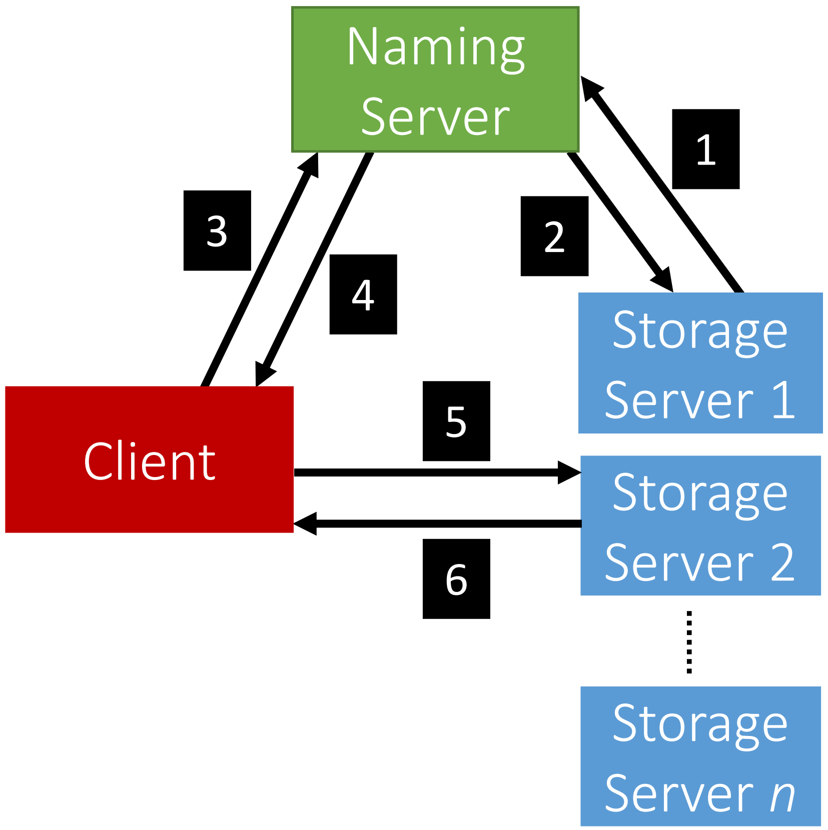
- Applied the knowledge of client-server communication and Remote Method Invocation (RMI) to build a Distributed File System denoted as FileStack

- Employed stubs and skeletons to mask communication, thereby transparently locating and manipulating files stored remotely at a cluster of machines

Entities & Architecture

- Storage Servers (SSs)
  - Each SS stores physically files to share in a directory (denoted as temporary directory) in its local file system
- Naming Server (NS)
  - Stores metadata about all shared files in the form of a mapping from filenames to storage servers (like DNS)
- Clients
  - Perform operations on files (e.g., write, read etc.)
- Architecture
  - Based on client-server architecture

Communication b/w Entities



Request-Reply Communication Paradigm

1. Registration
2. Duplicate Files, Create, Delete
3. CreateFile, CreateDirectory, IsDirectory, Delete, List, GetStorage
4. Results, Storage Stub
5. Read, Write, Size
6. Results (*of Read, Write, Size*)

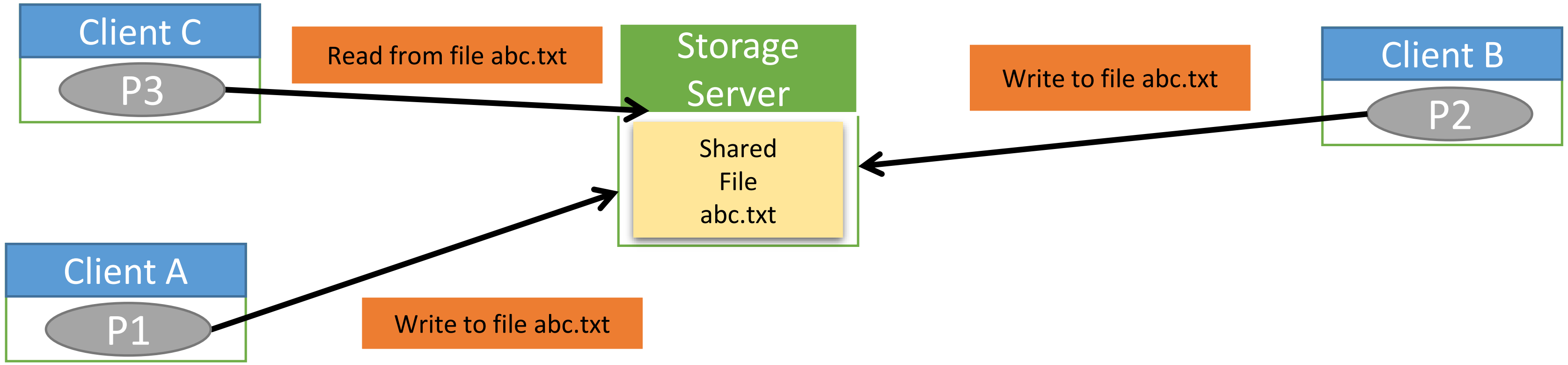
File Correctness & Consistency

- Did we allow multiple clients to write on a file?

  Yes!

- Did we allow a client to read a file under modification?

  Yes!



Project 2 Objectives

1. Devise and apply a synchronization algorithm that:
   - achieves *correctness* while sharing files
   - and ensures *fairness* to clients.

2. Devise and apply a replication algorithm that:
   - achieves load-balancing among storage servers
   - and ensures consistency of replicated files.

Project 2 Objectives

1. Logical Synchronization of Readers and Writers

2. Devise and apply a replication algorithm that:
   - achieves load-balancing among storage servers
   - and ensures consistency of replicated files.

Mutual Exclusion

1. Reader:
   - Reader is a Client who wishes to read a file at a SS
   - Reader first requests a read/non-exclusive/shared lock
2. Writer:
   - Writer is a Client who wishes to write to a file at a SS
   - Writer first requests a write/exclusive lock
3. Order:
   - Readers and writers are queued and served in the FIFO order

Read Locks

- Readers request the NS for read locks before reading files
- Readers do not modify contents of a file/directory
- Multiple readers can acquire a read lock simultaneously
- Readers unlock files once done

Write Locks

- Writers request the NS for write locks before reading/writing to files
- Writers can modify contents of files/directories
- Only one writer can acquire a write lock at a time
- Writers unlock files once done

Write Locks

- NS grants a write lock on a file if:
  - No reader is currently reading the file
  - No writer is currently writing to the file
- Assume a writer requests a write lock for project2.txt:

  /FileStack/users/student1/work/project2.txt

- NS applies read locks on all the directories in the path to prevent modifications
- NS then grants a write lock to the requestor of project2.txt

Service Interface

- Two new operations available to Clients:
  - LOCK(path, read/write)
  - UNLOCK(path, read/write)

Project 2 Objectives

1. Devise and apply a synchronization algorithm that:
   - achieves *correctness* while sharing files
   - and ensures *fairness* to clients.

2. Dynamic Replication of Files

Why Replicate?

- In our DFS, we'll have two kinds of Files:
  - Files that have a lot of requests
    - These are denoted as "***hot-files***"
  - Files that are very rarely accessed
    - These are denoted as "***cold-files***"
- To achieve load-balancing, we can replicate "***hot-files***" onto other SSs

How Many Replicas?

- To measure file how "hot" a file is, the NS can keep track of the number of requests to a file:
  - *num_requesters:* number of read requests to a file

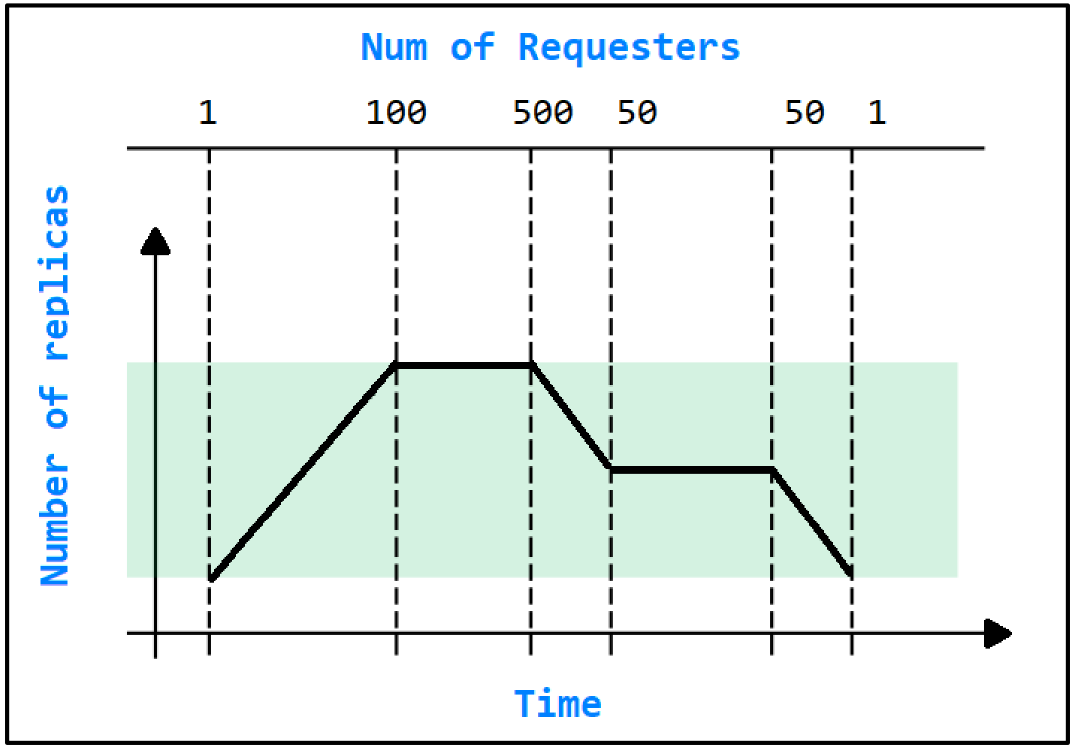
- *To scale replicas linearly with the increase of num_requests:*
  - *num_replicas* = α * *num_requesters*

How Many Replicas?

- However, we need to limit the number of replicas:
  - *num_replicas = min(α * num_requesters, upper_bound)*

- This is still too sensitive/fine-grained:
  - *num_requests_coarse: num_requests* rounded to the nearest multiple of 20
  - *num_replicas = min(α * num_requests_coarse, replica_upper_bound)*

How Many Replicas?



When to Replicate?

- NS would want to store *num_requests* as file metadata
- However, how can we determine and in turn update *num_requests* over time?
  - We know that Clients invoke read operations on storage servers
  - Therefore, every "read" lock request from a client is deemed as a read operation
  - Afterward, NS increments *num_requests*
  - Reavaluate num_replicas

How can we Replicate?

- NS first elects SSs to store the replicas
- NS commands each elected SS to copy the file from the original SS
- Therefore, the metadata of a file now includes *a set of SSs* instead of a single SS

How to Update Replicas

- **When a Client requests a write lock on a file:**
  - It causes the NS to *invalidate* all the replicas except the locked one
- Invalidation is achieved by **commanding those SSs hosting replicas to delete the file**
- When the Client unlocks the file, the NS commands SSs to copy the modified file

The Command Interface

- One new operation available to the NS:
    - Copy(path P, StorageStub S)

      copies file with path P from StorageStub S