Based on slides by Harsha V. Madhyastha

EECS 482 Introduction to Operating Systems

Spring/Summer 2020

Lecture 24: Distributed file systems

Nicole Hamilton
https://web.eecs.umich.edu/~nham/
nham@umich.edu

Agenda

1. Final exam.
2. Project 4.
3. Grading.
4. Student evals.
5. Distributed file systems.

Agenda

1. Final exam.
2. Project 4.
3. Grading.
4. Student evals.
5. Distributed file systems.

Final exam

Just like the midterm.

Online using *Crabster.org* Thu Aug 20 3:00 to 5:00 pm EDT.

We will post a link via Canvas and Piazza once the exam is live.

We will monitor Piazza for questions.

Material for the final:

1. Everything except my bonus lecture.

Final exam policy

The exam will be *"open everything"* except collaboration.

You can use any existing resource, including lecture notes, the book, your project solutions, you can even use Google, and your IDE.

The only thing you can't do is collaborate with others, including using social media to solicit help. If you can find an existing answer on stackexchange that's helpful, that's fair game. But you can't post a question.

Also, parts of the exam ask for short answers, which must be *in your own words*. Cutting and pasting word-for-word from an existing source and "close copying" will be treated as plagiarism and reported to the Honor Council.

Agenda

1. Final exam.
2. Project 4.
3. Grading.
4. Student evals.
5. Distributed file systems.

Project 4

No free space list or map is kept in the filesystem, so it has to be recreated each time you initialize by traversing the entire directory structure.

Could be a real problem with a multi-TB filesystem.

Alternatives:

1. Mark a shadow block as in use when allocated, even before it's linked into the directory. Plan to occasionally leak during crashes, run a utility to find lost space.
2. Use a logging scheme with commits.

Project 4 Testing

State space coverage

Test every request type with every possible state.

For example: FS_CREATE

File vs. directory.

In root directory vs. elsewhere.

Adding direntry in first data block vs. later.

Free direntry at the beginning vs. later.

Test close to resource limits.

Disk size, max path name, max file name, …

Project 4 Testing

Verifying concurrency

- Test with a pair of requests.
- Insert sleep( ) calls into your server so you can test race conditions.
- Consider every combination of request types.
- Vary commonality in pathnames.
- Block around “slow” operations.

Macro test cases

- Crash or deadlock with lots of concurrent requests.
- Check for memory and filesystem leaks.

Agenda

1. Final exam.
2. Project 4.
3. Grading.
4. Student evals.
5. Distributed file systems.

Grading

The isolation forced on us by the pandemic is hard on everyone but it’s especially hard on young people.

In a class like this, it’s hard to remain engaged in online-only lectures and it’s hard to effective and productive working on a team that’s only virtual.

The workload is also quite significant, more than in many classes.

This is the first time offered in the summer and could be hard to manage on top of an internship.

And with a smaller class, there was a much less active Piazza forum where you might pick up hints from answers to other people’s questions.

Grading

So, I know this is a really hard semester for some of you and you may be worried, if you didn't do well on P3, "Will I pass?"

I intend to give you a curve that is *at least as generous*, e.g., percentage of class receiving an A+ or an A, as in past semesters for 482.

I intend to do my best to pass everyone.

Agenda

1. Final exam.
2. Project 4.
3. Grading.
4. Student evals.
5. Distributed file systems.

Student evals

Please get them done.

They are super important.

The 370 and other questions …

If you wonder if they matter …

Agenda

1. Final exam.
2. Project 4.
3. Grading.
4. Student evals.
5. Distributed file systems.

Distributed file systems

Remote storage of data that appears local

Examples:

- Andrew File System (AFS)
- Dropbox
- Google Drive

Benefits?

- Share files across users.
- Uniform view of file system across machines.

Caching for performance

Bottleneck if many clients interacting with server?

- Server
- Network

Benefits of client-side caching:

- Improves server scalability.
- Better latency and throughput.
- Reduces network traffic.
- Can improve availability if the remote server crashes.

Client-side caching

Two approaches:

1. Migrate: Transfer sole copy from server to client.

   Simpler to implement. No need to keep copies in sync.

   Concurrent reads leads to ping-ponging as the sole copy keeps getting moved.

2. Replicate: Create additional copy at client.

   Clients can read from local copy.

   Must worry about inconsistent replicas.

How do the two approaches compare?

Handling writes with caching

If clients use write-back caching (writing dirty pages only when they're evicted) other clients may read stale data.

How to preserve consistency?

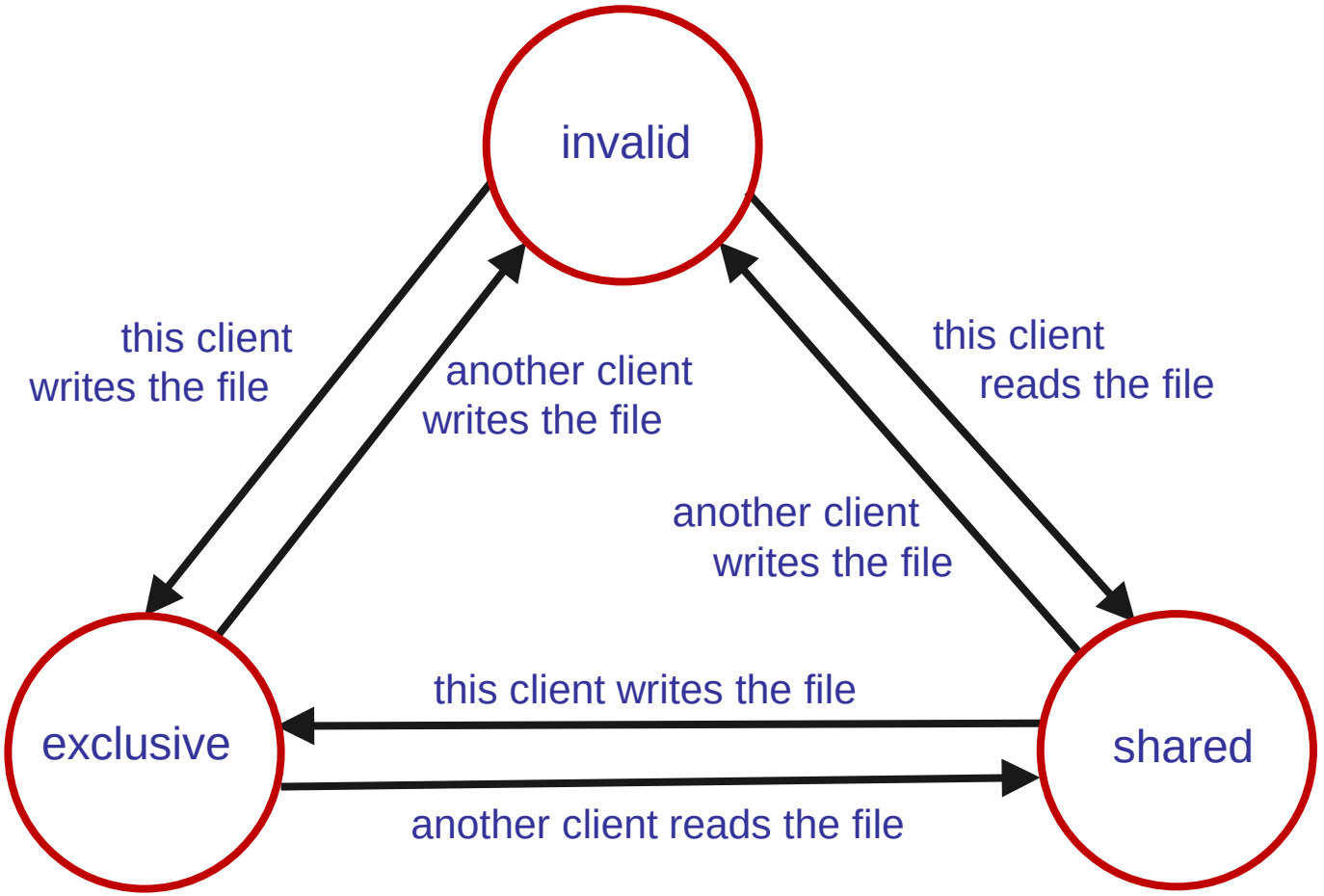
Write-through cache, updating the copy at server on *every* write?

Update **all** copies or invalidate **other** copies.

Pros and cons?

The invalidation message is probably a lot smaller than the copy and some copies may never be referenced again.

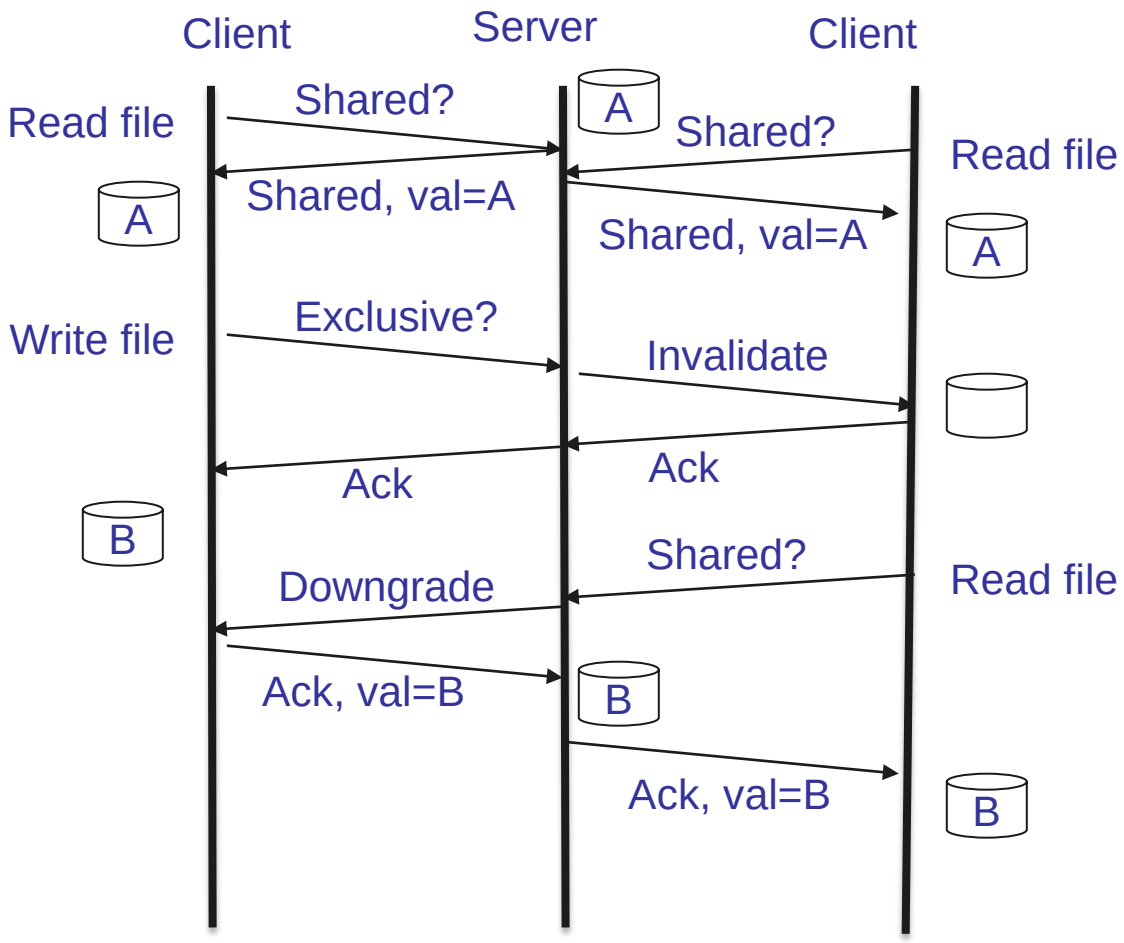
State machine for cached copy



Similar to anything else we've discussed previously?

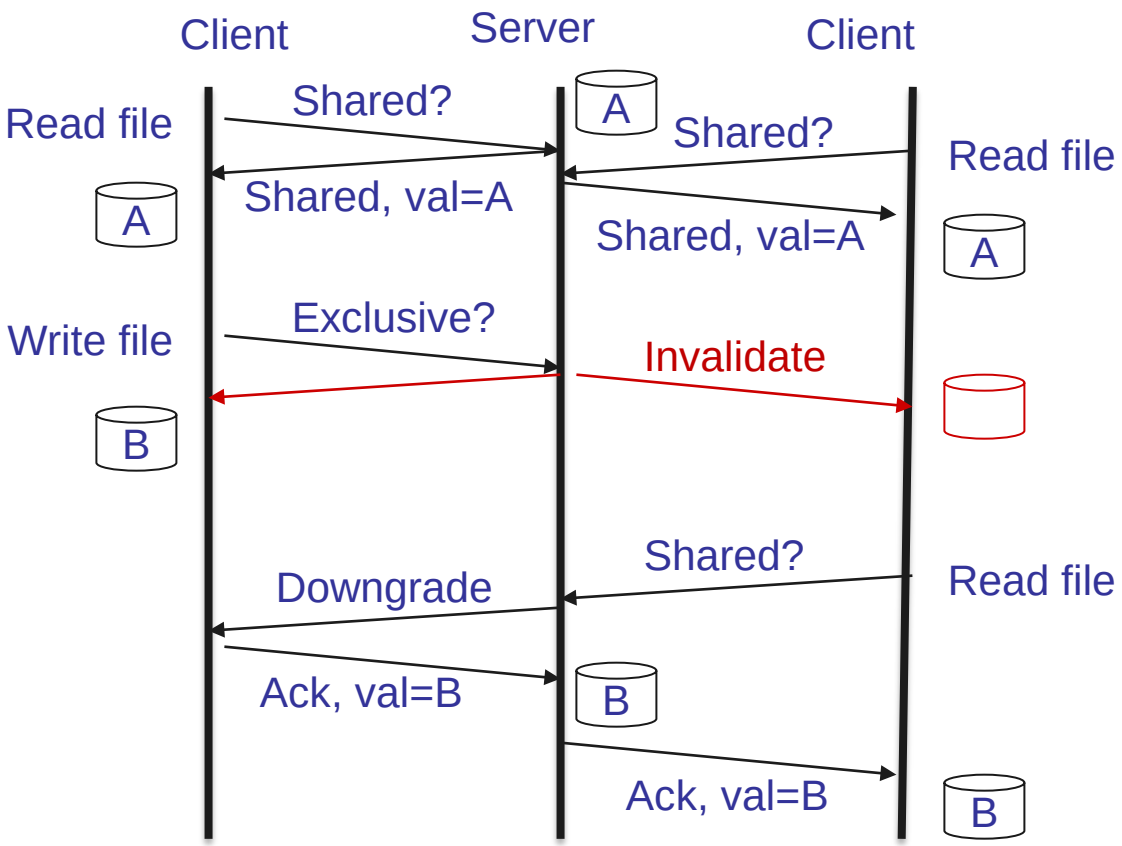
Invalidation protocol

Server waits for all the clients to acknowledge the invalidation before granting the exclusive use.



Order of operations

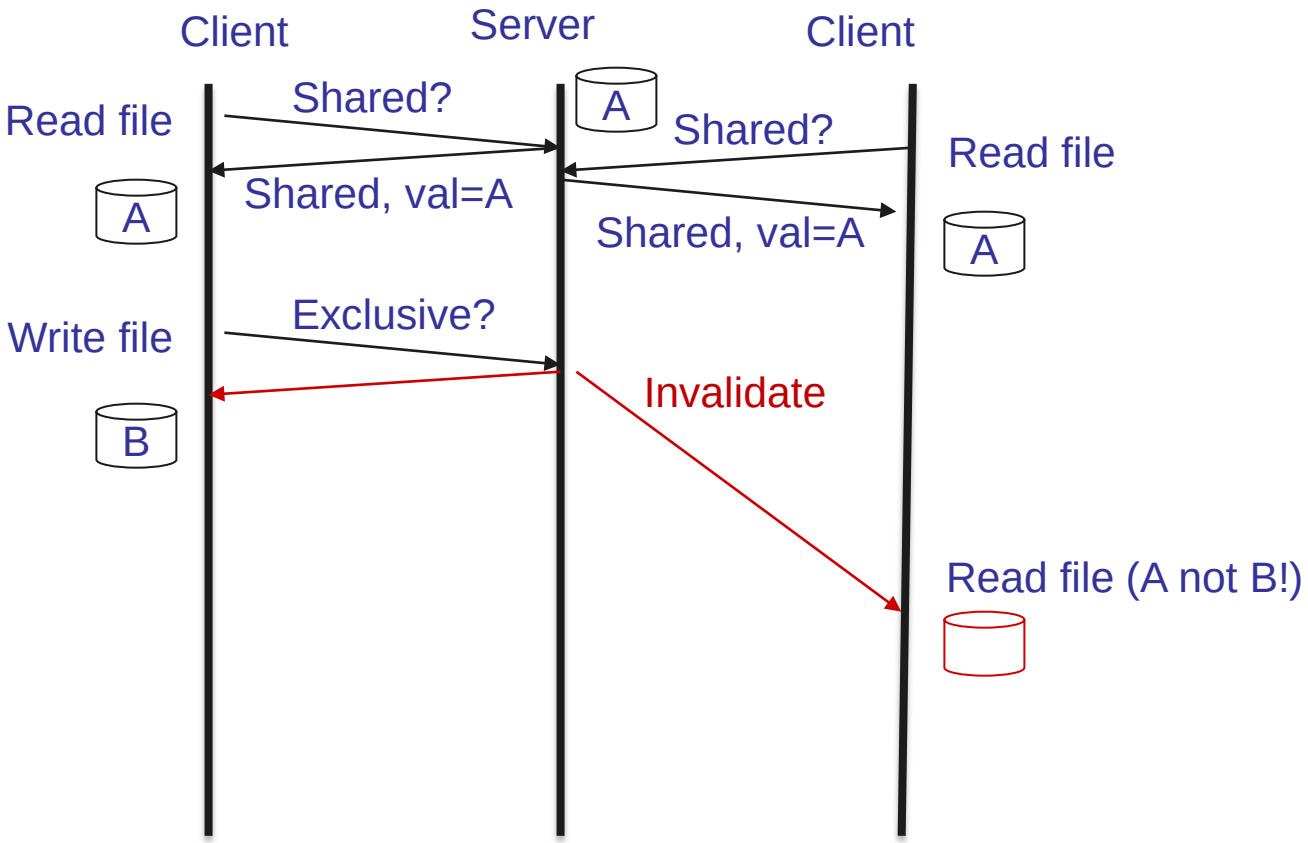
Is it necessary to wait for invalidations to be acknowledged?



Order of operations

Is it necessary to wait for invalidations to be acknowledged?

Yes, because the invalidation message may take a long time to get delivered.



Optimistic concurrency control

Alternate approach to prevent inconsistency.

- Allow clients to modify replicas freely.
- Server detects and resolves conflicting updates.

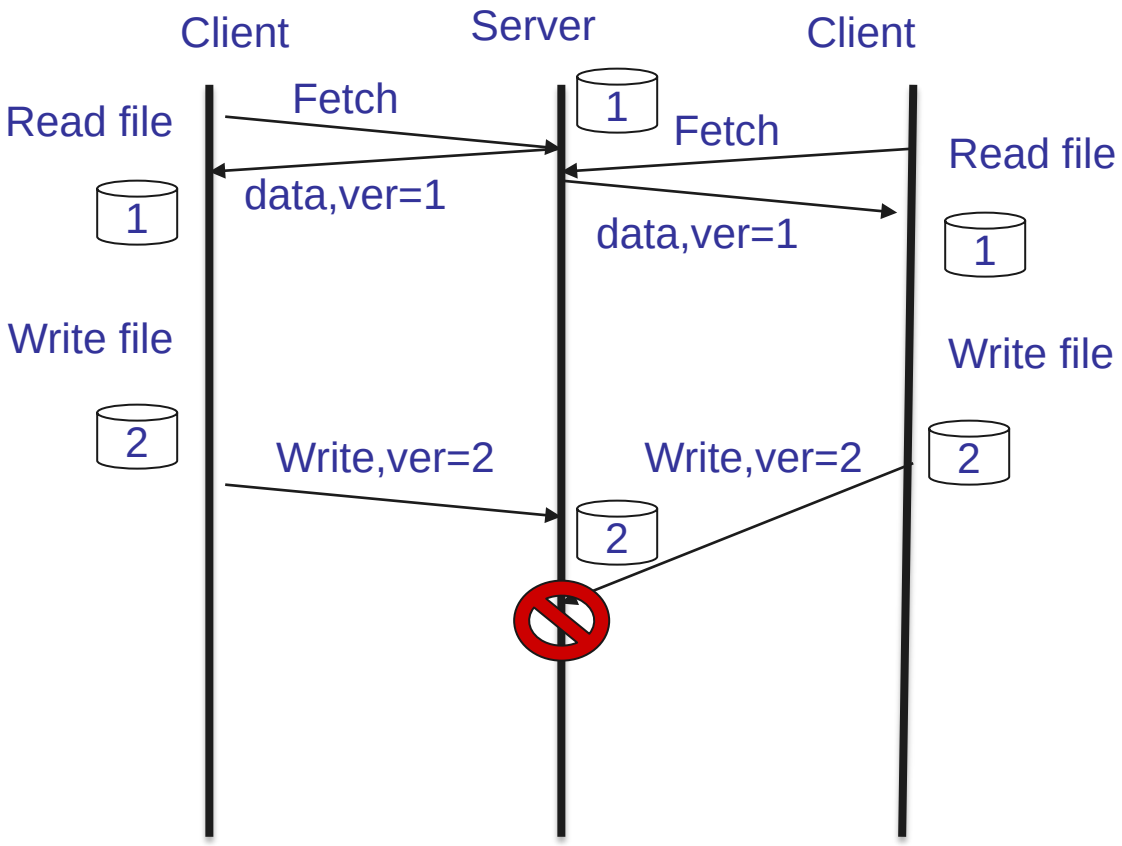
How to detect conflicts?

- Assign a version to each object (file, etc.)
- Increment the version on update.
- Conflict if server version >= client version.

Conflicting operations

Discover you have an old copy only when you try to write.

Means you could be reading old data but allows updating offline. (Dropbox model.)



Load balancing across servers

Two options:

1. Partitioning *clients* across servers not very sensible.
2. Partition *files* across servers makes more sense for sharing files.

How to find a file?

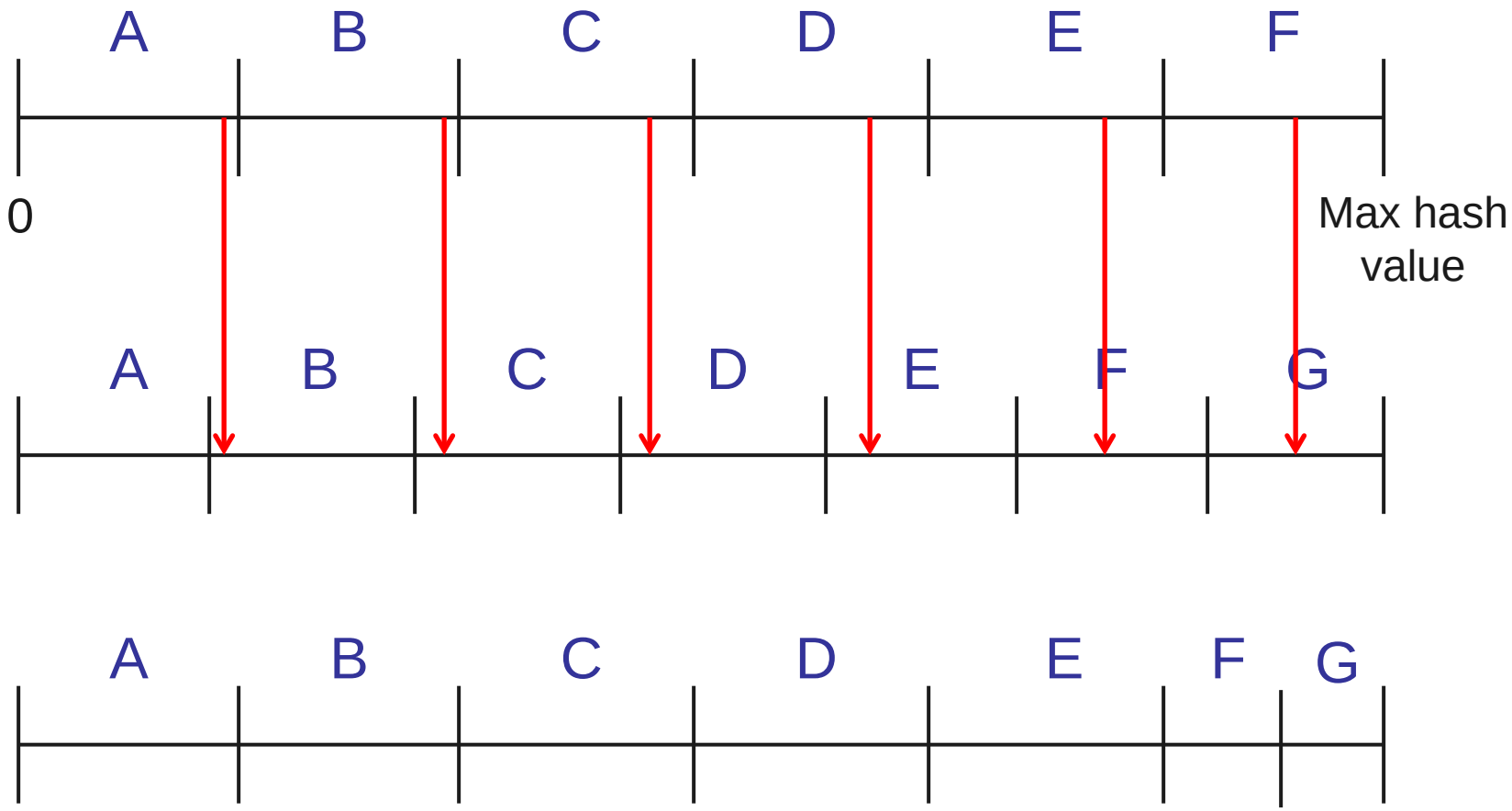
1. Ask a directory server.
2. Hash-based mapping:

   If N servers, store file *foo* on server *hash(foo) % N*

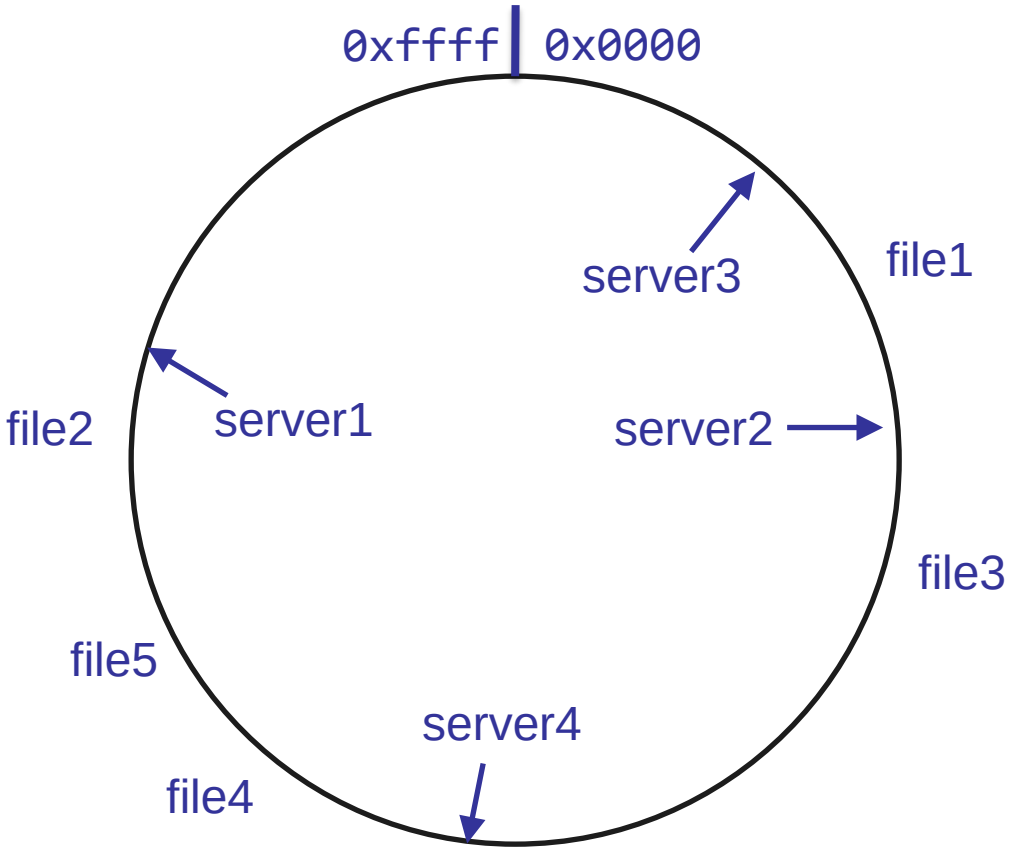
   What if we need to add or remove a server?

   File is now mapped to server *hash(foo) % (N+1)*

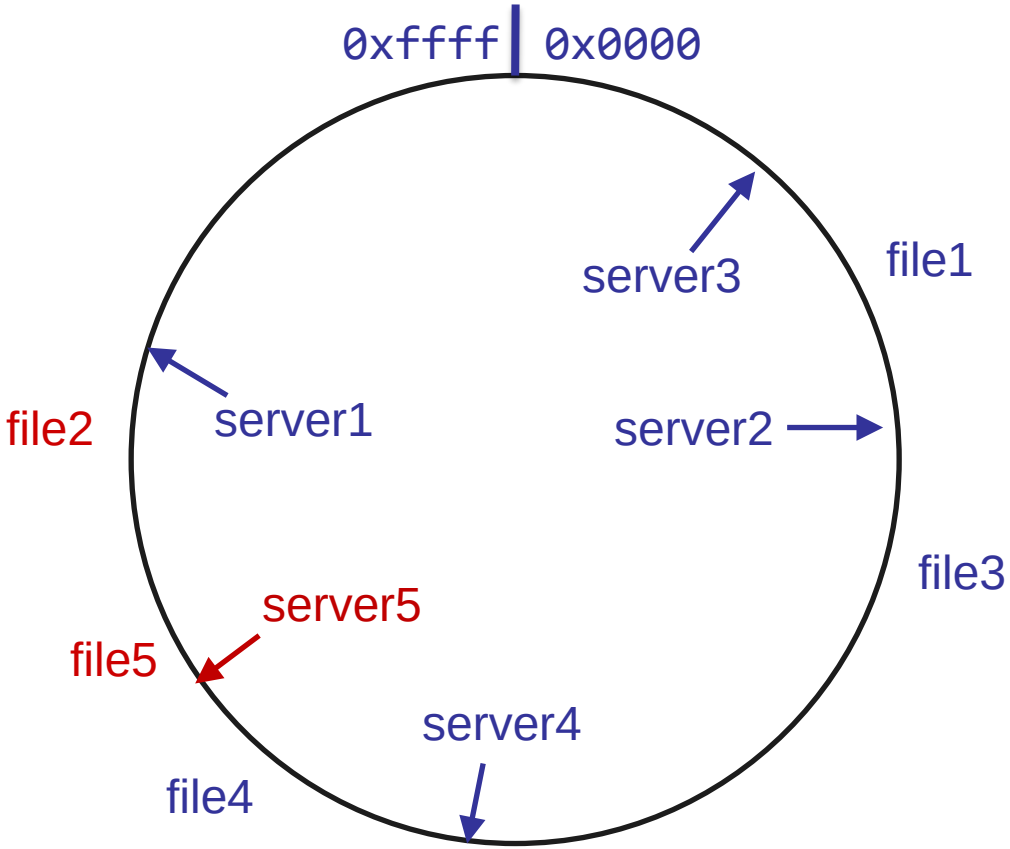
Load balancing across servers



Consistent hashing



Adding a server



Replication across servers

More servers → Increased likelihood of failure.

Replicate files to tolerate failures.

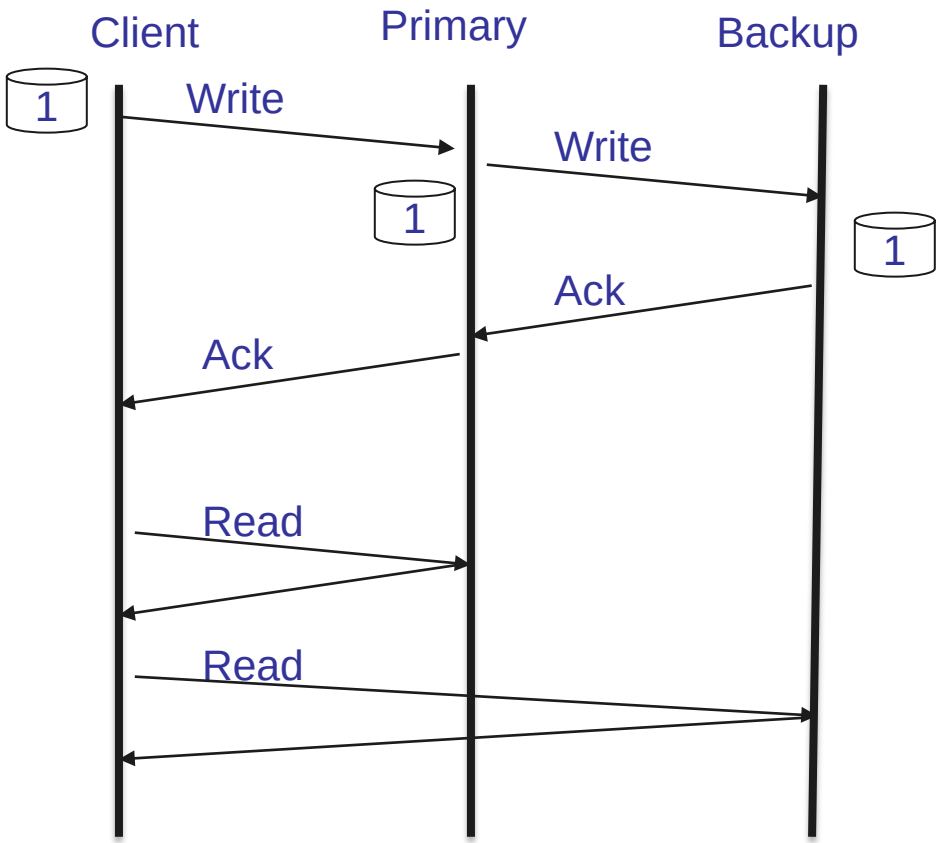
Example: Primary + backup

Write data by writing to both primary AND backup.

Read data by reading from primary OR backup.

Primary-Backup

If the primary fails, the client reads from the backup.



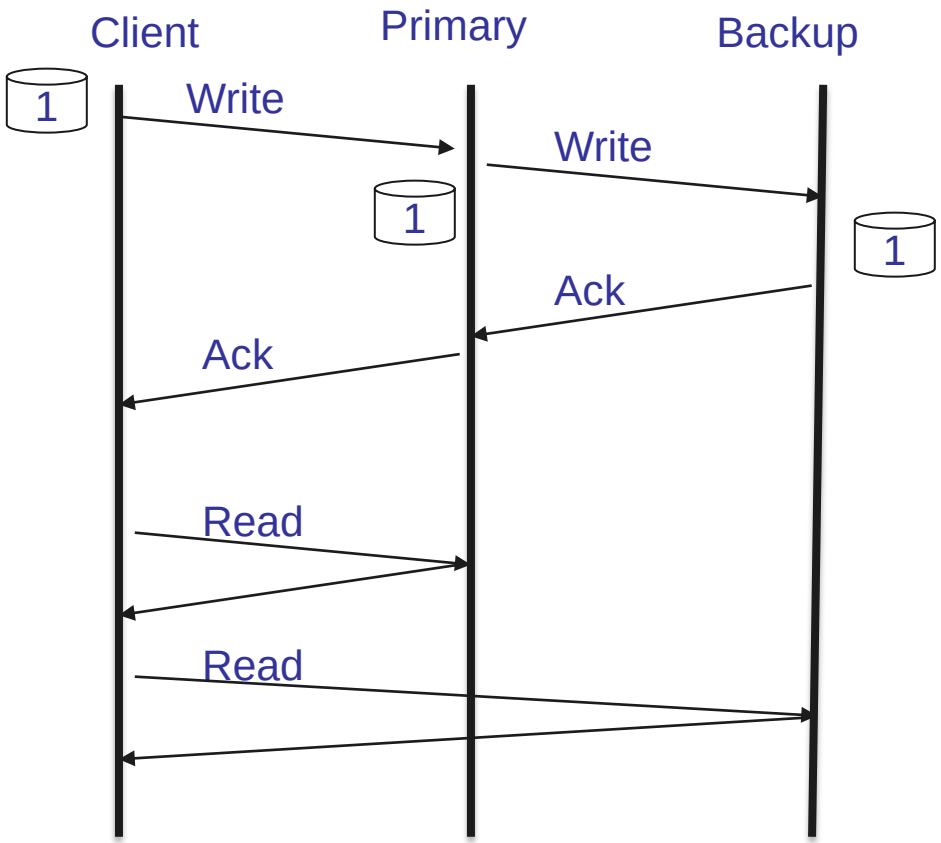
Primary-Backup

When primary fails

Backup becomes new primary.

Backup handles all reads/writes.

Primary recovers, syncs state, can become backup.

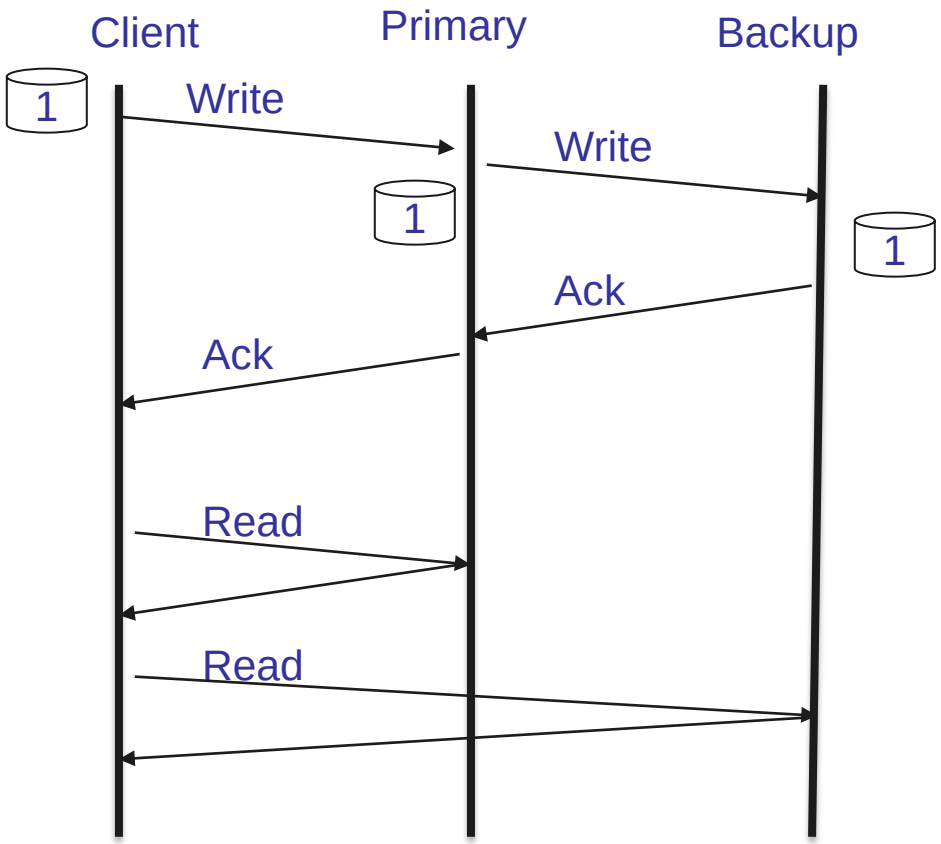


Primary-Backup

When backup fails

Primary handles all reads/writes.

Backup recovers, syncs state.



Fault tolerance

What if backup fails before primary recovers?

Data is unavailable, lost if failures permanent.

How can we tolerate 2 failures?

Use 2 backups.

Need f+1 servers to tolerate f failures.

Common practice is 3 copies but geographically separated.

What are we assuming about failures?

Fault models

Fail stop (primary-backup)

- Machine stops executing immediately.
- Can detect the failure.
- Examples: power failure, OS crash.

Byzantine

- Anything goes!
- Server may send erroneous messages.
- Server could be malicious/hacked.
- Example: Servers differ on file contents. Which is correct?

Byzantine generals

All loyal generals decide upon the same plan of action.

Traitor may do anything they wish.

A small number of traitors cannot cause the loyal generals to adopt a bad plan.

Source: Lamport et al, “The Byzantine Generals Problem”, https://people.eecs.berkeley.edu/~luca/cs174/byzantine.pdf

Byzantine generals

Say there's one commander C and two lieutenants L1 and L2.

Goal: Decide whether to attack enemy or retreat.

Attack succeeds if at least 2 attack.

1 of the 3 is a traitor.

Naïve approach: L1 and L2 follow C's command to either attack or retreat.

Solution: C sends command to L1 and L2, who then exchange notes and follow majority.

Byzantine generals

C sends command to L1 and L2, who then exchange notes and follow majority

Case 1: L1 is traitor

C sends attack to both L1 and L2

L2 receives {attack, retreat}

Case 2: C is traitor

C sends attack to L1 and retreat to L2

L1 receives {attack, retreat}

Byzantine generals

Need at least 4 generals to cope with 1 traitor.

3f+1 generals to cope with f traitors.

Solution: C sends command to L1, L2, and L3, who then exchange notes and follow majority.

Three cases:

C sends 3 attacks.

C sends 2 attacks, 1 retreat.

C sends 1 attack, 2 retreats.