Primary Backup

Tom Anderson

Last Time

- MapReduce
  - Fault tolerance
  - Discussion
- RPC
  - At least once
  - At most once
  - Exactly once

Topics

- Primary/backup replication
  - Example of state machine replication
- Lab 2 Hints

Primary/Backup

- Many clients, sending requests concurrently to service
- Two server roles: primary, backup
  - Which server is which changes due to failures
- Lab 2: viewservice defines which server is primary, which is backup

Primary-Backup Goals

- A service that behaves like a single, always available server:
  - available: service still usable despite [some] server and network failures
  - correct: acts like a single server to clients
- Lab 2: key-value service
  - Simple operations: Get, Put, PutHash

State Machine Replication

- Replicate state on a set of servers
- Apply operations (RPCs) in the same order on each server
  - view each replica as a state machine
  - each replica starts in the same state
  - applies all operations in the same order
  - -> they end up in the same state
- Key assumption: operations are deterministic

Primary/Backup In Operation

- Clients send operations to primary
  - Put, PutHash, Get
- Primary chooses the order of operations to apply
- Primary forwards sequence of operations to backup
- Backup performs operations in same order (hot standby)
  - Or just saves log of operations (cold standby)
- Primary replies to client
  - After backup has saved the operation order

Challenges

- Non-deterministic operations
  - Examples? Solutions?
- Network failures and partitions
- State transfer between primary and backup
  - after backup becomes primary, need a new backup
- There can be only one primary at a time
  - Need to keep clients, primary, and backup in sync as to who is primary and who is backup

View Service

View server decides who is primary and backup
- Clients and servers depend on the view server
- They do NOT decide on their own (why not?)

The tricky part:
- Only one primary at a time!
- But clients may not have gotten the message!
- Careful protocol design needed

View service assumed to never fail
- Fixed in Lab 3

Repair

- Primary fails
- View server declares new “view”, promotes backup to primary
- View server promotes “idle” server as new backup
- Primary initializes new backup's state
- Now ok if new primary fails

Example

View server maintains a sequence of "views”

- view #, primary, backup

1: S1 S2

2: S2 –

3: S2 S3

View Server

View server monitors server liveness

Each server periodically sends a Ping RPC to view server

- "dead" if missed N Pings in a row
- "live" after single Ping

Can server be alive but declared “dead” by viewserver? Why?

View Server

Any number of servers can Ping view server

- if more than two, the others are "idle" servers

If primary is dead

- new view with previous backup as primary

If backup is dead, or no backup

- new view with previously idle server as backup

OK to have a view with a primary and no backup

- Why?

Question

How can we ensure new primary has up-to-date replica of state?

- only promote previous backup
- don't make a previously idle server the primary (except at startup)

What if the backup hasn't had time to acquire the state?

Avoid promoting a state-less backup?

1: S1 S2

S1 stops pinging viewserver

2: S2 S3

S2 *immediately* stops pinging

3: S3 –

Question

How does viewserver know if backup is up to date?

- Primary must ack new view (once backup is up to date)
- View server stays with current view until ack
- Even if primary has failed (or appears to have failed)

Question

Can more than one server think it is primary at the same time?

More Than One Primary

1: S1, S2

- Network breaks, so viewserver thinks S1 dead
- But S1 is still alive

2: S2, --

- But S1 alive and not aware of view #2, so S1 still thinks it is primary
- AND S2 alive and thinks it is primary

- Called “split brain”

More Than One Primary

Can we ensure only one server acts as primary?

- Even though more than one may *think* it is primary
- Acts as = responds to client requests

1: S1 S2

2: S2 --

- S1 still thinks it is primary
- S1 must forward operations to S2 (for backup!)
- S2 thinks S2 is primary
- S2 can reject S1's forwarded operations

Rules

1. Primary in view i must have been primary or backup in view i-1
2. Primary must wait for backup to accept/execute each request before doing operation and replying to client
3. Non-backup must reject forwarded requests
   - Backup accepts forwarded requests only if they are in the current view
4. Non-primary must reject direct client requests
5. Every operation must be before or after state transfer

Example

1: S1, S2
- View server stops hearing Pings from S1

2: S2, --
- It may be a while before S2 hears about view #2

Before S2 hears about view #2:
- S1 can process ops from clients, S2 will accept forwarded ops
- S2 will reject ops from clients who have heard about view #2 [but trigger a fetch to get new view from view server]

After S2 hears:
- if S1 receives client op, it will forward but S2 will reject
- S1 will send error to client, client will ask view server for new view
- Client will re-send op to S2

State Transfer

How does a new backup get the current state?
- e.g. the contents of the key/value store
- If S2 is backup in view i, but was not in view i-1
- S2 asks primary to transfer the state

Rules for state transfer:
- every op must be either before or after state xfer
- if op before xfer, xfer must reflect op
- if xfer before op, primary forwards op after xfer finishes

Progress

Are there cases when the protocol cannot make forward progress?

Fast Reads

- Does the primary need to forward read (Get) requests to the backup, or can it reply directly?

Three Approaches To State Transfer (Key-Value Store)

1. Transfer current k-v map
   - Between operations! All RPCs appear to occur either before or after the transfer.
   - Recall that "at most once" means you need to save the response you previously gave to an RPC. When backup takes over, needs to act as if it was the primary.
   - If each client has only one RPC outstanding at a time, state = map + result of last RPC from each client.
2. State can be derived from an operation log: the sequence of RPC's processed at the primary.
   - Simpler, less efficient
   - Current value of key is last value written in the log for that key
3. Transfer checkpoint plus log of recent changes to k-v map

Another Implementation Hint

In Section, transition diagram for the view server for Lab 2a. What about Lab 2b?

- State transition diagram for primary?
- State transition diagram for backup?
- State transition diagram for client?
- All of the above plus the viewserver?

Another Implementation Hint

Behavior of a distributed system is the cross-product of the states of the individual nodes

- state at the clients
- state at the primary and backup
- state at the view server

Transitions are message send/receive and local events

Many possible paths are allowable -- depending on which messages are delivered in which order, which nodes do which actions

Lamport Clocks

Can we make sure everyone agrees on the same order of events?

An issue if:

- multiple clients, multiple servers
- even if there are no failures
- even if messages are delivered in order sent by each client ("processor order")

Facebook Storage System

Initially:

- a few front end web servers to do application logic
- a single backend storage server

To scale, add more front ends, more back end servers:

- Each front end pulls data from multiple servers (e.g., one for privacy settings, one for pictures).
- Do users see a consistent view?

Now add some intermediate caches:

- 100+ lookups per page
- 1B+ users: 1M+ front ends, 1M+ caches, 1M+ servers

Example: Arranging Lunch

Example: Shared Whiteboard

Example: Parallel Make

Physical Clocks

- Can we assign every event in a distributed system a unique wall clock time stamp?
- Local clocks aren't perfect
  - Crystals oscillate at slightly different frequencies
  - Typical error is ~ 2 seconds/month
- Synchronize clocks across distributed system?
  - Network messages involve delays
  - Network message delays are variable

Physical Clocks

- Lets assume a network-attached GPS
  - How close can we bound clocks across multiple systems?
- Option 1: client polls the GPS server for current time.
  - How far off will the timestamp be when it arrives back at the client?
- Option 2: repeatedly fetch the GPS time, estimate relative rate of skew of the local clock

Logical Clocks (Centralized implementation)

Send every message to a central arbiter, which assigns an order for all messages.

Problems with centralization?

Space-Time Diagram