RPC and Clocks

Tom Anderson

Last Time

- Go
  - Synchronization
  - RPC
- Lab 1 RPC

Topics

- MapReduce
  - Fault tolerance
  - Discussion
- RPC
  - At least once
  - At most once
  - Exactly once
- Lamport Clocks
  - Motivation

MapReduce Fault Tolerance Model

Master is not fault tolerant

- Assumption: this single machine won't fail while running a mapreduce app

Many workers, so have to handle their failures

- Assumption: workers are fail stop
- They can fail and stop
- They may reboot
- They don't send garbled weird packets after a failure

What kinds of faults does MapReduce need to tolerate?

- Network:
  - lost packets
  - duplicated packets
  - temporary network failure
  - server disconnected
  - network partitioned
- Worker:
  - crash+restart
  - Permanent failure
  - All workers fail simultaneously -- power/earthquake
  - Crash mid-way through complex operation
- What if?
  - Bug in map function, so that mapper crashes every time?

Tools for Dealing With Faults

- Retry
  - if pkt is lost: resend
  - worker crash: give task to another worker
  - may execute MR job twice! (is this ok? Why?)
- Replicate
  - E.g., input files on multiple storage servers
- Replace
  - E.g., add new worker after old one fails

Lab 1 MapReduce Simplifications

- No key in map
- Assume global file system
- No partial failures
  - Files either completely written or not created
  - If restart some failed operation, ok to write to the same filename

DeWitt/Stonebraker Critique

- A giant step backward in the programming paradigm for large-scale data intensive applications
- A sub-optimal implementation, in that it uses brute force instead of indexing
- Not novel at all: represents a specific implementation of well known techniques developed nearly 25 years ago
- Missing most of the features that are routinely included in current DBMS
- Incompatible with all of the tools DBMS users have come to depend on

To understand why some technologies win:

The Innovator's Dilemma, Clayton Christensen

Remote Procedure Call (RPC)

A request from the client to execute a function on the server.

- On client
  - Ex: z = DoMap(worker, i)
  - Parameters marshalled into a message (can be arbitrary types)
  - Message sent to server (can be multiple pkts)
  - Wait for reply
- On server
  - message is parsed
  - operation (DoMap(i)) invoked
  - Result marshalled into a message (can be multiple pkts)
  - Message sent to client

RPC vs. Procedure Call

- What is equivalent of:
  - The name of the procedure?
  - The calling convention?
  - The return value?
  - The return address?

RPC vs. Procedure Call

Binding
- Client needs a connection to server
- Server must implement the required function
- What if the server is running a different version of the code?

Performance
- procedure call: maybe 10 cycles = ~3 ns
- RPC in data center: 10 microseconds => ~1K slower
- RPC in the wide area: millions of times slower

Failures
- What happens if messages get dropped?
- What if client crashes?
- What if server crashes?
- What if server appears to crash but is slow?
- What if network partitions?

Three Options if RPC Doesn't Return

- At least once (NFS, DNS, …)
  - keep retrying until RPC succeeds
- At most once (Go, …)
  - Retry, but make sure RPC is never executed more than once
- Exactly once
  - Make sure RPC is always executed and never executed more than once

At Least Once

RPC library waits for response for a while

If none arrives, re-send the request

Do this a few times

Still no response -- return an error to the application

Non-replicated key/value server

Client sends Put(a)

Server gets request, but network drops reply

Client sends Put(a) again

- should server respond "yes"?
- or "no"?

What if operation is "deduct $10 from bank account"?

Does TCP Fix This?

- TCP: reliable bi-directional byte stream between two endpoints
  - Retransmission of lost packets
  - Duplicate detection
- But what if TCP times out and client reconnects?
  - Browser connects to Amazon
  - RPC to purchase book
  - Wifi times out during RPC
  - Browser reconnects

When is at-least-once OK?

- If no side effects
  - read-only operations (or idempotent ops)
- If application has its own plan for detecting duplicates

- Example: NFS
  - readFileBlock
  - writeFileBlock

At Most Once

Client includes unique ID (UID) with each request
- use same UID for re-send

Server RPC code detects duplicate requests
- return previous reply instead of re-running handler

```
if seen[uid] {
    r = old[uid]
} else {
    r = handler()
    old[uid] = r
    seen[uid] = true
}
```

Some At-Most-Once Issues

How do we ensure UID is unique?

- Big random number?
- Combine unique client ID (IP address?) with sequence #?
- What if client crashes and restarts? Can it reuse the same UID?
- Maybe client should get its unique ID from the server?

When Can Server to Discard Old RPCs?

Option 1:
  Never?
Option 2:
  unique client IDs
  per-client RPC sequence numbers
  client includes "seen all replies <= X" with every RPC
Option 3: only allow client one outstanding RPC at a time
  arrival of seq+1 allows server to discard all <= seq
Option 4: client agrees to keep retrying for < 5 minutes
  server discards after 5+ minutes

What if Server Crashes?

If at-most-once list of recent RPC results is stored in memory, server will forget and accept duplicate requests when it reboots

- Does server need to write the recent RPC results to disk?
- If replicated, does replica also need to store recent RPC results?

Go RPC is "at most once" and "usually once"

- Open TCP connection
- Write request to TCP connection
- TCP may retransmit, but server's TCP will filter out duplicates
- No retry in Go code (i.e. will NOT create 2nd TCP connection)
- Go RPC code returns an error if it doesn't get a reply
  - perhaps after a timeout (from TCP)
  - perhaps server didn't see request
  - perhaps server processed request but server failed before reply came back
  - Perhaps server processed request and network failed

Go RPC at-most-once is not enough

What if RPC sent over TCP, but reply never arrives and socket fails?

- If worker doesn't respond, the master re-sends to another worker
- But original worker may have not failed, and is working on it too

Go RPC can't detect this kind of duplicate

- In lab 2 you will have to protect against these kinds of duplicates

Exactly Once

To survive client crashes, client needs to record pending RPC's on disk

- So that we can replay them with the same UID

To survive server crashes, need to record results of completed RPC's on disk

- So that we can suppress duplicates

In other words, similar to two phase commit!

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