Lecture 13:
- Distributed File Storage: GFS (cont)
- Intro to Map Reduce

Announcements:
- R-2 out
- HW-2 out

GFS Architecture: Overview

The main GFS components:
- **GFS Master**: single centralized control, one per cluster
- **GFS Chunkserver**: one per machine, manages file partitions (“chunks”)
- **GFS Client**: library running on each application (app can be in cluster)
GFS Architecture: Data Append

**Two cases for “data mutations” updates vs append**

1. *Update*: client has filename, byte range, buffer of write data
2. *Append*: client has filename, buffer of data to append

**Basic Idea: Append**

- Each chunk replica is mutated ... default: 3 replicas per chunk
- Master selects one **primary** replica ... others become **secondary** replicas
- The primary is granted a 60 second **lease** ... can be extended if needed
- During the lease, primary manages chunk changes
- Primary picks a serial order for all chunk mutations
- The secondary replicas follow the same order when applying mutations

---

GFS Architecture: Record Append

**Append Process:** Assume $P$ is the primary, $S$ is a secondary

1. Client asks master for $P$ and $S$’s ... assigns $P$ (and lease) if needed
2. Master replies with $P$ and $S$’s ... cached for future mutations
3. Client sends data to all replicas ... pipelined to closest, who forwards, etc.
4. After client receives receipt, sends write request to $P$  
   
   (*)

5. $P$ forwards request to $S$’s, who write data in same order as $P$  
   
   (†)

6. $S$’s reply to $P$ indicating they completed mutation
7. $P$ replies to client either success or error ...

(*) $P$ orders all changes it receives (across clients), applies changes in order
(†) $P$ also tells $S$’s where to write the data in the chunk

© S. Bowers
GFS Architecture: Record Append

Replica Write Failures:

- If one or more replicas fail, writes “stick” at $P$ and non-failing $S$’s
- Client retries steps 3–7 a few times, then retries write with master
- A successful retry can lead to repeated data at some replicas
- Note that all append operations are at the same offset across replicas
- On successful append, client is sent the offset of appended data

Example of appending data $X$, $Y$, and $Z$ with replica failure

Guarantees and Non-Guarantees:

- Between appended data, could be padding or append-data duplicates
- The in-between regions are considered inconsistent
- Clients see same regions resulting from sequence of successful mutations
- Replicas versioned by master when granting leases (for available replicas)
- Because clients cache chunk locations, can read from stale replicas
- Concurrent non-append writes not guaranteed to be serializable
MapReduce: Intro

MapReduce Overview:

A programming model for simplifying data-intensive computing

• computations split into a “Map” phase and a “Reduce” phase
• programmers implement Map and Reduce interfaces

MapReduce framework then handles the overall execution

• distribution/replication, parallelization, data movement, and failures

MapReduce led to the open-source Apache Hadoop system

• similar ideas also still used for data-intensive systems
• e.g., in serverless, auto-scaling, “no-ops” systems

Inspired by map and reduce functions in functional programming:

map : function, list → list ... takes a unary function

  e.g.: map(f, [a, b, c, ⋯]) ⇒ [f(a), f(b), f(c), ⋯]
  easily parallelizable! (with enough resources)

reduce : function, list → value ... takes a binary function

  • e.g.: reduce(+, [1, 2, 3, ⋯]) ⇒ (⋯((1+2) + 3) + ⋯)
  • similar to fold – which implements an accumulator pattern

Map and Reduce interfaces are similar in spirit, but with some differences ...
Summary – Things to Know

• Understand basic process of mutations (including record append) in GFS

• Understand basic consistency guarantees and non-guarantees in GFS and reason behind them

• Understand basic idea of MapReduce