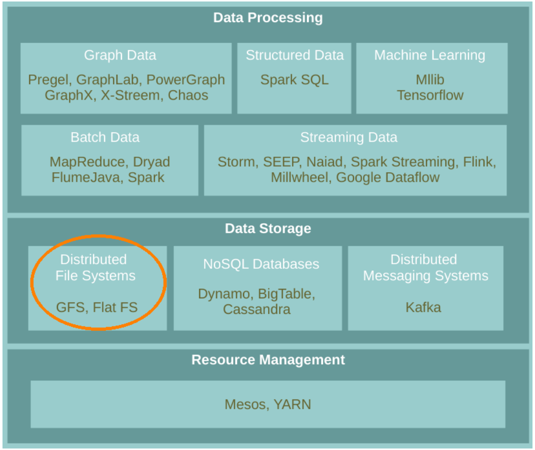
Large Scale File Systems

Amir H. Payberah
payberah@kth.se
30/08/2019

The Course Web Page

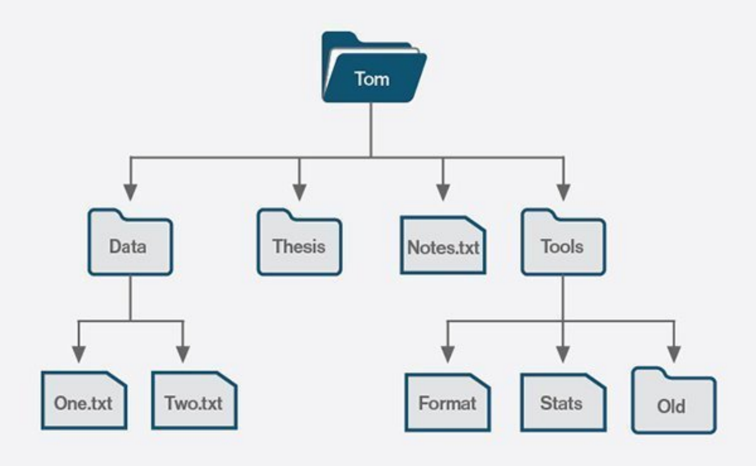
https://id2221kth.github.io

Where Are We?


Data Processing
Graph Data
Pregel, GraphLab, PowerGraph
GraphX, X-Streem, Chaos
Structured Data
Spark SQL
Machine Learning
Mllib
Tensorflow
Batch Data
MapReduce, Dryad
FlumeJava, Spark
Streaming Data
Storm, SEEP, Naiad, Spark Streaming, Flink,
Millwheel, Google Dataflow
Data Storage
Distributed
File Systems
GFS, Flat FS
NoSQL Databases
Dynamo, BigTable,
Cassandra
Distributed
Messaging Systems
Kafka
Resource Management
Mesos, YARN

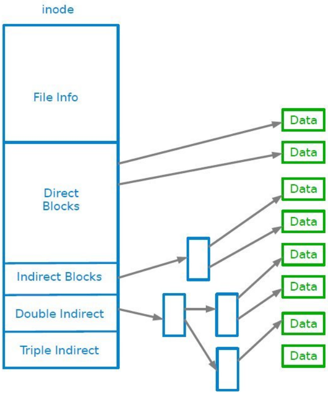
File System

What is a File System?



What is a File System?

- Controls how data is stored in and retrieved from disk.



Distributed File Systems

- When data outgrows the storage capacity of a single machine: partition it across a number of separate machines.
- Distributed filesystems: manage the storage across a network of machines.

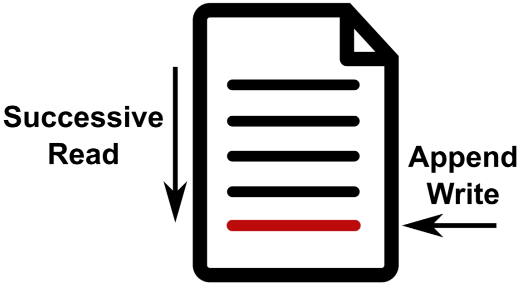
Google File System (GFS)

Motivation and Assumptions

- Huge files (multi-GB)
- Most files are modified by appending at the end
  - Random writes (and overwrites) are practically non-existent
- Optimise for streaming access
- Node failures happen frequently

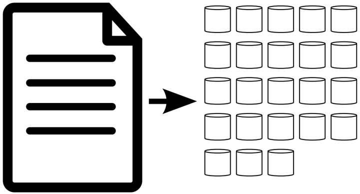
Optimised for Streaming

- Write once, read many.

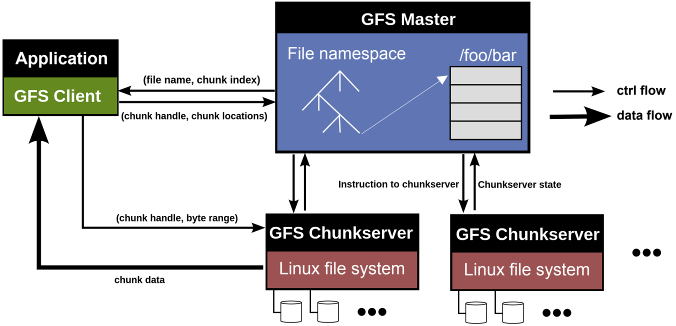


Files and Chunks

- Files are split into chunks.
- Chunks, single unit of storage.
  - Immutable
  - Transparent to user
  - Each chunk is stored as a plain Linux file

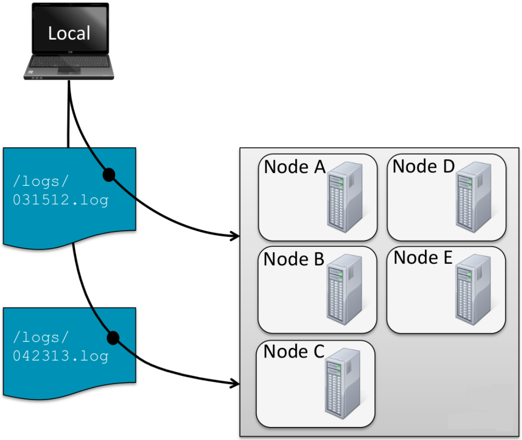


GFS Architecture

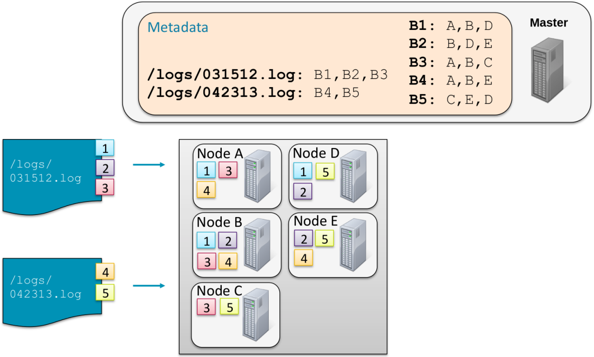


- Main components:
  - GFS master
  - GFS chunk server
  - GFS client

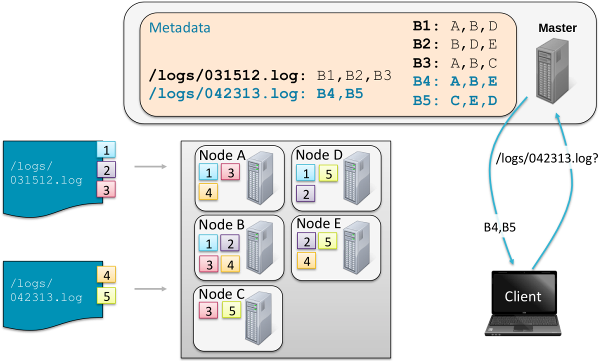
Big Picture - Storing and Retrieving Files (1/4)



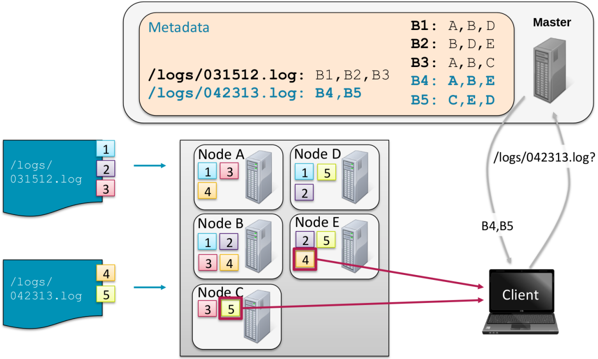
Big Picture - Storing and Retrieving Files (2/4)



Big Picture - Storing and Retrieving Files (3/4)

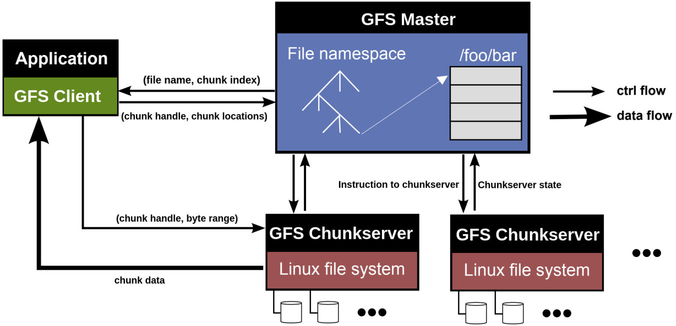


Big Picture - Storing and Retrieving Files (4/4)



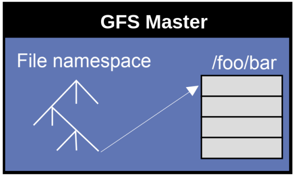
System Architecture Details

GFS Architecture



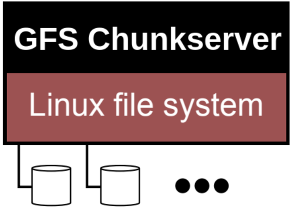
GFS Master

- Responsible for all system-wide activities
- Maintains all file system metadata
  - Namespaces, ACLs, mappings from files to chunks, and current locations of chunks
  - All kept in memory, namespaces and file-to-chunk mappings are also stored persistently in operation log
- Periodically communicates with each chunkserver
  - Determine chunk locations
  - Assesses state of the overall system



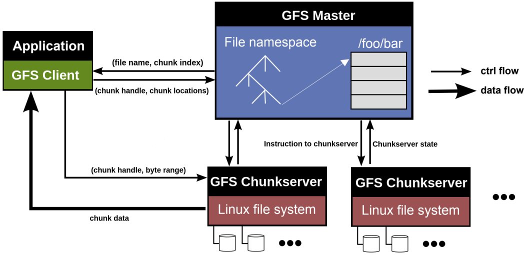
GFS Chunk Server

- Manage chunks
- Tells master what chunks it has
- Store chunks as files
- Maintain data consistency of chunks



GFS Client

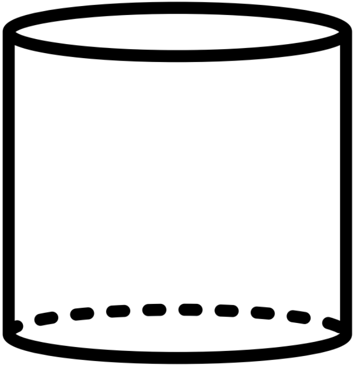
- Issues control requests to master server.
- Issues data requests directly to chunk servers.
- Caches metadata.
- Does not cache data.



Data Flow and Control Flow

- Data flow is decoupled from control flow
- Clients interact with the master for metadata operations (control flow)
- Clients interact directly with chunkservers for all files operations (data flow)

Why Large Chunks?



Why Large Chunks?

- 64MB or 128MB (much larger than most file systems)
- Advantages
  - Reduces the size of the metadata stored in master
  - Reduces clients need to interact with master
- Disadvantages
  - Wasted space due to internal fragmentation

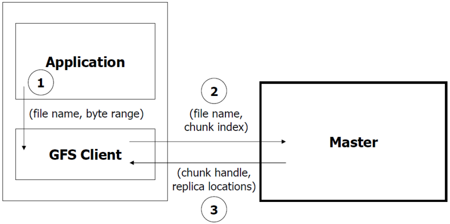
System Interactions

The System Interface

- Not POSIX-compliant, but supports typical file system operations
  - `create`, `delete`, `open`, `close`, `read`, and `write`
- `snapshot`: creates a copy of a file or a directory tree at low cost
- `append`: allow multiple clients to append data to the same file concurrently

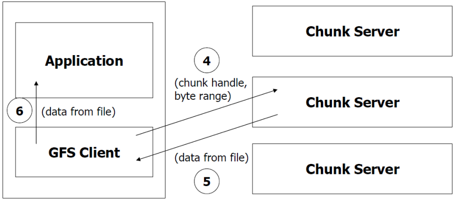
Read Operation (1/2)

- 1. Application originates the read request.
- 2. GFS client translates request and sends it to the master.
- 3. The master responds with chunk handle and replica locations.



Read Operation (2/2)

- 4. The client picks a location and sends the request.
- 5. The chunk server sends requested data to the client.
- 6. The client forwards the data to the application.



Update Order (1/2)

- Update (mutation): an operation that changes the content or metadata of a chunk.
- For consistency, updates to each chunk must be ordered in the same way at the different chunk replicas.
- Consistency means that replicas will end up with the same version of the data and not diverge.

Update Order (2/2)

- For this reason, for each chunk, one replica is designated as the primary.
- The other replicas are designated as secondaries
- Primary defines the update order.
- All secondaries follows this order.

Primary Leases (1/2)

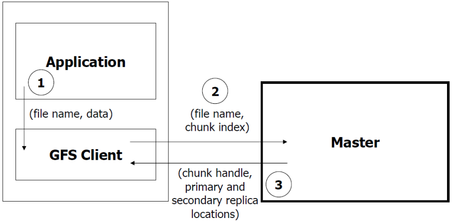
- For correctness there needs to be one single primary for each chunk.
- At any time, at most one server is primary for each chunk.
- Master selects a chunk-server and grants it lease for a chunk.

Primary Leases (2/2)

- The chunk-server holds the lease for a period T after it gets it, and behaves as primary during this period.
- If master does not hear from primary chunk-server for a period, it gives the lease to someone else.

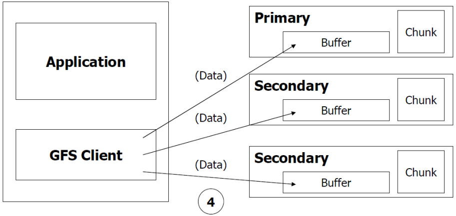
Write Operation (1/3)

- 1. Application originates the request.
- 2. The GFS client translates request and sends it to the master.
- 3. The master responds with chunk handle and replica locations.



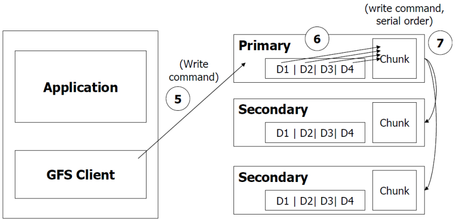
Write Operation (2/3)

- 4. The client pushes write data to all locations. Data is stored in chunk-server's internal buffers.



Write Operation (3/3)

- 5. The client sends write command to the primary.
- 6. The primary determines serial order for data instances in its buffer and writes the instances in that order to the chunk.
- 7. The primary sends the serial order to the secondaries and tells them to perform the write.



Write Consistency

- Primary enforces one update order across all replicas for concurrent writes.
- It also waits until a write finishes at the other replicas before it replies.
- Therefore:
  - We will have identical replicas.
  - But, file region may end up containing mingled fragments from different clients: e.g., writes to different chunks may be ordered differently by their different primary chunk-servers
  - Thus, writes are consistent but undefined state in GFS.

Append Operation (1/2)

- 1. Application originates record append request.
- 2. The client translates request and sends it to the master.
- 3. The master responds with chunk handle and replica locations.
- 4. The client pushes write data to all locations.

Append Operation (2/2)

- 5. The primary checks if record fits in specified chunk.
- 6. If record does not fit, then the primary:
  - Pads the chunk,
  - Tells secondaries to do the same,
  - And informs the client.
  - The client then retries the append with the next chunk.
- 7. If record fits, then the primary:
  - Appends the record,
  - Tells secondaries to do the same,
  - Receives responses from secondaries,
  - And sends final response to the client

Delete Operation

- Meta data operation.
- Renames file to special name.
- After certain time, deletes the actual chunks.
- Supports undelete for limited time.
- Actual lazy garbage collection.

The Master Operations

A Single Master

- The master has a global knowledge of the whole system
- It simplifies the design
- The master is (hopefully) never the bottleneck
  - Clients never read and write file data through the master
  - Client only requests from master which chunkservers to talk to
  - Further reads of the same chunk do not involve the master

The Master Operations

- Namespace management and locking
- Replica placement
- Creating, re-replicating and re-balancing replicas
- Garbage collection
- Stale replica detection

Namespace Management and Locking (1/2)

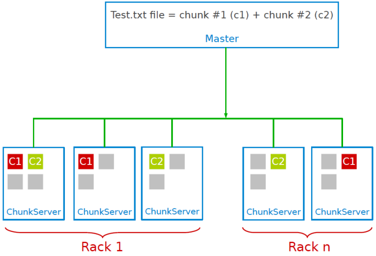
- Represents its namespace as a lookup table mapping pathnames to metadata.
- Each master operation acquires a set of locks before it runs.
- Read lock on internal nodes, and read/write lock on the leaf.
- Example: creating multiple files (`f1` and `f2`) in the same directory (`/home/user/`).
  - Each operation acquires a read lock on the directory name `/home/user/`
  - Each operation acquires a write lock on the file name `f1` and `f2`

Namespace Management and Locking (2/2)

- Read lock on directory (e.g., `/home/user/`) prevents its deletion, renaming or snapshot
- Allowed concurrent mutations in the same directory

Replica Placement

- Maximize data reliability, availability and bandwidth utilization.
- Replicas spread across machines and racks, for example:
  - 1st replica on the local rack.
  - 2nd replica on the local rack but different machine.
  - 3rd replica on the different rack.
- The master determines replica placement.



Creation, Re-replication and Re-balancing

- Creation
  - Place new replicas on chunk servers with below-average disk usage.
  - Limit number of recent creations on each chunk servers.

- Re-replication
  - When number of available replicas falls below a user-specified goal.

- Rebalancing
  - Periodically, for better disk utilization and load balancing.
  - Distribution of replicas is analyzed.

Garbage Collection

- File deletion logged by master.
- File renamed to a hidden name with deletion timestamp.
- Master regularly deletes files older than 3 days (configurable).
- Until then, hidden file can be read and undeleted.
- When a hidden file is removed, its in-memory metadata is erased.

Stale Replica Detection

- Chunk replicas may become stale: if a chunk server fails and misses mutations to the chunk while it is down.
- Need to distinguish between up-to-date and stale replicas.
- Chunk version number:
  - Increased when master grants new lease on the chunk.
  - Not increased if replica is unavailable.
- Stale replicas deleted by master in regular garbage collection.

Fault Tolerance

Fault Tolerance for Chunks

- Chunks replication (re-replication and re-balancing)
- Data integrity
  - Checksum for each chunk divided into 64KB blocks.
  - Checksum is checked every time an application reads the data.

Fault Tolerance for Chunk Server

- All chunks are versioned.
- Version number updated when a new lease is granted.
- Chunks with old versions are not served and are deleted.

Fault Tolerance for Master

- Master state replicated for reliability on multiple machines.
- When master fails:
  - It can restart almost instantly.
  - A new master process is started elsewhere.
- Shadow (not mirror) master provides only read-only access to file system when primary master is down.

GFS and HDFS

GFS vs. HDFS

GFS	HDFS
Master	Namenode
ChunkServer	DataNode
Operation Log	Journal, Edit Log
Chunk	Block
Random file writes possible	Only append is possible
Multiple write/reader model	Single write/multiple reader model
Default chunk size: 64MB	Default chunk size: 128MB

HDFS Example (1/2)

```
# Create a new directory /kth on HDFS
hdfs dfs -mkdir /kth
```

```
# Create a file, call it big, on your local filesystem and
# upload it to HDFS under /kth
hdfs dfs -put big /kth
```

```
# View the content of /kth directory
hdfs dfs -ls big /kth
```

```
# Determine the size of big on HDFS
hdfs dfs -du -h /kth/big
```

```
# Print the first 5 lines to screen from big on HDFS
hdfs dfs -cat /kth/big | head -n 5
```

HDFS Example (2/2)

```
# Copy big to /big hdfscopy on HDFS
hdfs dfs -cp /kth/big /kth/big_hdfscopy
```

```
# Copy big back to local filesystem and name it big_localcopy
hdfs dfs -get /kth/big big_localcopy
```

```
# Check the entire HDFS filesystem for problems
hdfs fsck /
```

```
# Delete big from HDFS
hdfs dfs -rm /kth/big
```

```
# Delete /kth directory from HDFS
hdfs dfs -rm -r /kth
```

Summary

Summary

- Google File System (GFS)
- Files and chunks
- GFS architecture: master, chunk servers, client
- GFS interactions: read and update (write and update record)
- Master operations: metadata management, replica placement and garbage collection

References

- S. Ghemawat et al., The Google file system, Vol. 37. No. 5. ACM, 2003.

Questions?