Distributed FileSystems: GFS

Google File System (GFS)

- Example of clustered file system
- Basis of Hadoop's and Bigtable's underlying file system
- Many other implementations

Design Constraints

- Motivating application: Google
- Component failures are the norm
  - 1000s of components: inexpensive servers and clients
  - Bugs, human errors, failures of memory, disk, connectors, networking, and power supplies
  - Constant monitoring, error detection, fault tolerance, automatic recovery are integral to the system
- Files are huge by traditional standards
  - Multi-GB files are common; each file contains application objects such as web documents
  - Billions of objects

Design Constraints

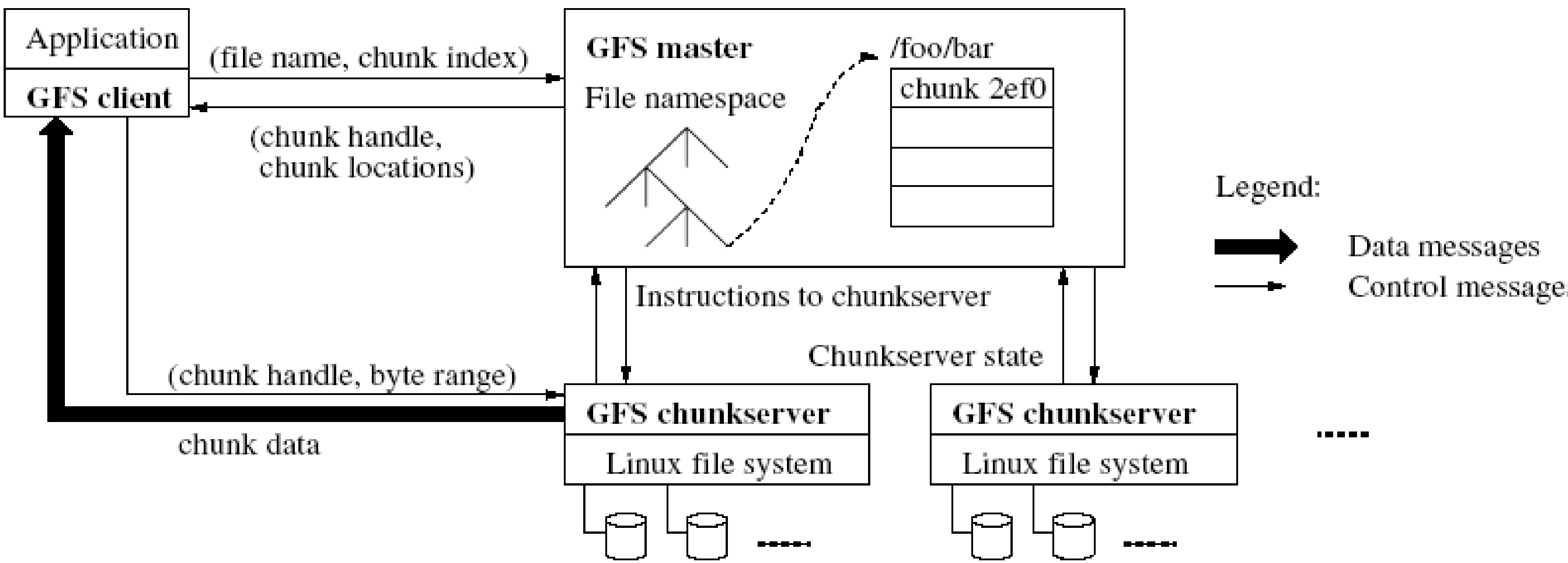
- Most modifications are appends
  - Random writes are practically nonexistent
  - Many files are written once, and read sequentially
- Types of reads
  - Data Analysis Programs reading large repositories
  - Large streaming reads (read once)
  - Small random reads (in the forward direction)
- Sustained bandwidth more important than latency

Interface Design

- Familiar file system interface
  - Create, delete, open, close, read, write
- Files are organized hierarchically in directories and identified by path names
- Additional operations
  - Snapshot (for cloning files or directories)
  - Record append by multiple clients concurrently guaranteeing atomicity but without locking

Architectural Design

- A GFS cluster
  - A single master
  - Multiple chunkservers per master
    - Accessed by multiple clients
  - Running on commodity Linux machines
- A file
  - Represented as fixed-sized *chunks*
    - Labeled with 64-bit unique and immutable global IDs
    - Stored at chunkservers on local disks as linux files
    - Replicated across chunkservers for reliability


Application
GFS client
(file name, chunk index)
(chunk handle,
chunk locations)
GFS master
File namespace
/foo/bar
chunk 2ef0
Legend:
Data messages
Control messages
Instructions to chunkserver
Chunkserver state
(chunk handle, byte range)
chunk data
GFS chunkserver
Linux file system
GFS chunkserver
Linux file system

- Master server
  - Maintains all metadata
    - Name space, access control, file-to-chunk mappings
  - Chunk lease management
  - Garbage collection, chunk migration
  - Sending Heartbeat messages to chunk servers
    - Giving instructions; Collection of state information

- GFS clients
  - Linked to applications
  - Communicates with master and chunkservers on behalf of the client
    - Consult master for metadata
    - Access data from chunkservers
  - No caching of file data at clients and chunkservers
    - Streaming reads and append writes do not benefit from caching

Single-Master Design

- Can use global knowledge to devise efficient decisions for chunk placement and replication
- Clients ask Master which chunkserver it should contact
  - This information is cached at the client for some time
  - A client typically asks for multiple chunk locations in a single request
- The master proactively provides chunk locations immediately following those requested
  - Reduces future interactions at no cost

Chunk Size

- 64 MB; much larger than typical file system block size
- Fewer chunk location requests to the master
  - Good for applications that mostly read and write large files sequentially
- Reduced network overhead to access a chunk
  - May keep a persistent TCP connection for some time
- Fewer metadata entries
  - Kept in memory
- Some potential problems with fragmentation

Metadata

- Three major types
  - File and chunk namespaces
  - File-to-chunk mappings
  - Locations of a chunk's replicas
- All kept in memory
  - The first two metadata are also kept persistent
  - Quick global scans
    - Garbage collections
    - Reorganizations for chunk failures, balancing load and better disk space utilization
  - 64 bytes per 64 MB of data

Chunk Locations

- No persistent states
  - Polls chunkservers at startup
  - Use heartbeat messages to monitor chunk servers thereafter
  - On-demand approach vs. coordination
    - On-demand wins when changes (failures) are often
      - Chunkservers join and leave a cluster
      - Chunkservers may change names, fail or restart
      - Changes happen often with a cluster having hundreds of servers

Operation Logs

- Metadata updates are logged
  - e.g., <old value, new value> pairs
  - Log replicated on remote machines
- Files and chunks, and their versions are also identified by the logical times at which they are created
- Take global snapshots (checkpoints) to truncate logs
- Recovery
  - Latest checkpoint + subsequent log files

System Interactions

- The master grants a chunk lease to a replica, called the primary
- The primary picks a serial order for all mutations to the chunk
  - A mutation is an operation that changes the contents or the metadata of a chunk
    - Write or append operation
- The replica holding the lease determines the order of updates to all replicas
- Lease
  - 60 second timeouts
  - Can be extended indefinitely by the primary as long as the chunk is being changed
  - Extension request are piggybacked on heartbeat messages
  - After a timeout expires, the master can grant new leases

Write Process

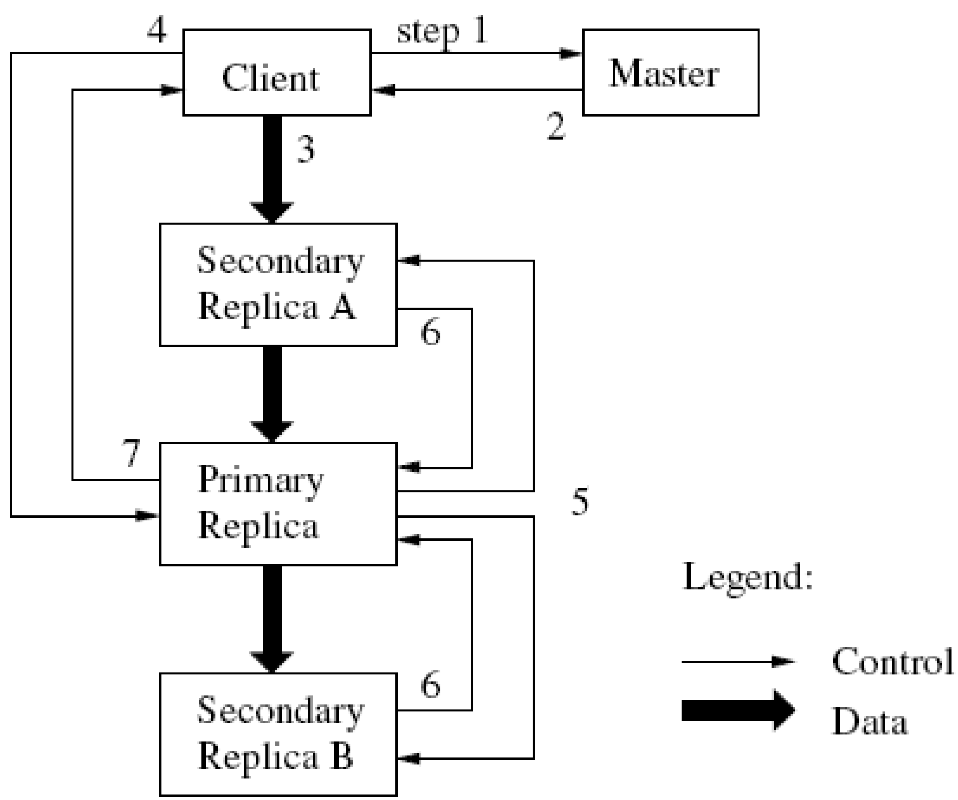


Figure 2: Write Control and Data Flow

Consistency Model

- Changes to namespace (i.e., metadata) are atomic
  - Done by single master server
  - Master uses log to define global total order of namespace-changing operations
- Data changes more complicated
- File region is consistent if all clients see as same, regardless of replicas they read from
- File region is defined after data mutation if it is consistent, and all clients see that entire mutation

- Serial writes guarantee region is defined and consistent
- But multiple writes from the same client may be interleaved or overwritten by concurrent operations from other clients
  - Consistent but not defined
- Record append completes at least once, at offset of GFS' choosing
  - Application must cope with this semantics, and possible duplicates

Data Flow

- Separation of control and data flows
  - Avoid network bottleneck
- Control flows from the client to the primary and then to all secondaries
- Data are pushed linearly among chain of chunkservers having replicas
  - Each machine forwards data to its closest machine which has not received it
  - Pipelined transfers using a switched networks with full duplex links

Master Operations

- Namespace management with locking
- Replica creation, re-replication
- Garbage Collection
- Stale chunk detection

Locking Operations

- Many master operations can be time consuming
  - Allow multiple master operations and use locks over regions of the namespace
  - GFS does not have a per directory data structure
  - GFS does not allow hard or symbolic links
  - Represents namespace as a lookup table mapping full pathnames to metadata.
- A lock per path
  - To access /d1/d2/leaf, need to lock /d1, /d1/d2, and /d1/d2/leaf
  - Can modify a directory concurrently
    - Each thread acquires a read lock on a directory and a write lock on a file
  - Totally ordered (first by level and lexicographically in the same level) locking to prevent deadlocks

Replica Placement

- Goals:
  - Maximize data reliability and availability
  - Maximize network bandwidth
- Need to spread chunk replicas across machines and racks
  - Guards against disk or machine failures (different machines)
  - Guards against network switch failures (different racks)
  - Utilizes aggregate network bandwidth for read operations
  - Write traffic has to flow through different racks
- Higher priority to replica chunks with lower replication factors
- Limited resources spent on replication

Replica creation, re-replication

- Creation of empty replicas
  - Placement in servers with below average disk utilization
  - Limit recent creation on the same server
  - Spread replicas across racks
- Re-replication
  - Server becomes unavailable or corrupted
  - Re-replicate chunks with some priority
    - Chunks having one replica
    - Chunks with live usage with running applications
    - Master is involved only in picking up a high priority chunk and then instructs a suitable server for cloning directly from the valid replica

Garbage Collection

- Deleted files are marked and hidden for three days
- Then they are garbage collected
  - Master deletes the metadata for the deleted file
  - Server identifies the set of deleted chunks during regular heartbeat messages
  - Server deletes its deletes replicas of chunks
- Combined with other background operations (regular scans of namespaces or handshakes with servers)
  - Done in batches and thus the cost is amortized
- Simpler than eager deletion due to
  - Unfinished replicated creation
  - Lost deletion messages
- Safety net against accidental irreversible deletions

Fault Tolerance and Diagnosis

- Fast recovery
  - Master and chunkserver are designed to restore their states and start in seconds regardless of termination conditions
    - No distinction between normal and abnormal termination
- Chunk replication
- Master replication
  - Master state is replicated on multiple machines
    - Operation log and checkpoints are replicated
    - Commit to a mutation happens after flushing logs to all replicas
  - Infrastucture outside GFS starts a new master on failure
    - Clients use a canonical name which is a DNC-alias for a server
  - Shadow masters provide read-only access when the primary master is down

Fault Tolerance: Data Integrity

- A chunk is divided into 64-KB blocks having 32bit checksum
- Each chunkserver uses checksum to detect corruption of stored data
- Verified at read and write times
  - Chunkserver returns error to the requestor and reports the error to master
    - Master creates a new replica and instructs to the server to delete its chunk
- Checksum has little performance overhead on read operations and on record append operations
- Chunkservers employ background scans for rarely used chunks

GFS: Summary

- Success: used actively by Google to support search service and other applications
  - Availability and recoverability on cheap hardware
  - High throughput by decoupling control and data
  - Supports massive data sets and concurrent appends
- Semantics not transparent to apps
  - Must verify file contents to avoid inconsistent regions, repeated appends (at-least-once semantics)
- Performance not good for all apps