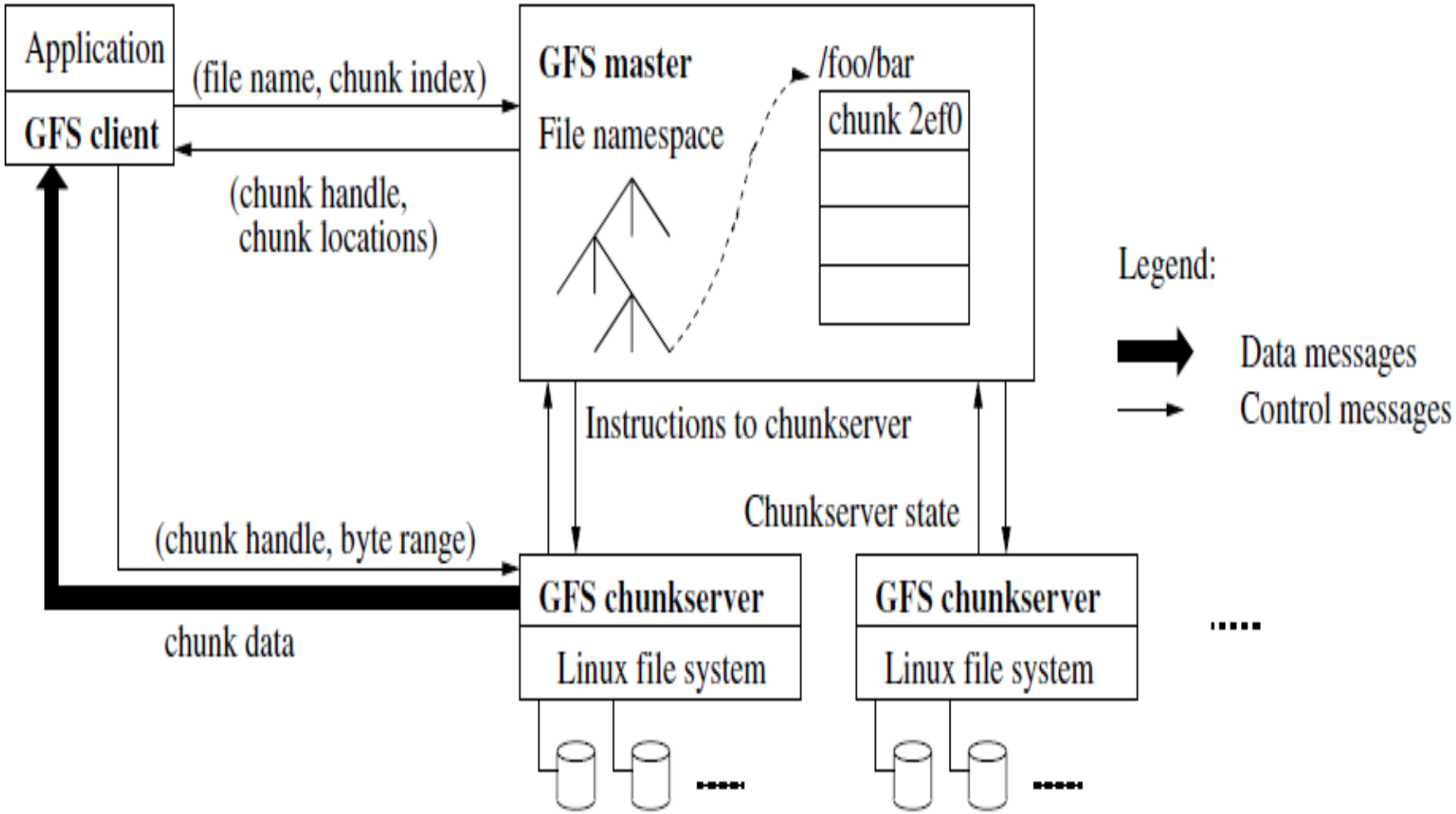
Google File System

Dr. Amit Praseed

Assumptions behind GFS

- Component failures are the norm rather than the exception
- Files are huge by traditional standards
- Common workloads encountered
  - large streaming reads and small random reads
  - large, sequential writes that append data to files
- System must efficiently implement well-defined semantics for multiple clients that concurrently append to the same file
- High sustained bandwidth is more important than low latency

GFS Architecture



GFS Cluster Organization

- GFS cluster has
  - Single master
  - Multiple chunk servers
- Files are divided into fixed size chunks
  - Default 64 MB size (Reasonably huge – why?)
  - Identified by a global 64 bit chunk handle
  - Each chunk replicated on multiple chunk servers (by default 3)

Operations of the Master Server

- Maintains file system metadata
  - Namespace
  - Access control information
  - Files ←→ chunks mapping
  - Chunk locations etc.
- Other bookkeeping tasks
  - Chunk lease management
  - Garbage collection
  - Chunk migration
- Checks chunk server availability and health
  - Heartbeat messages

Mutations and Leases

- A mutation is an operation that changes the contents or metadata of a chunk
  - write or an append operation.
- Each mutation is performed at all the chunk's replicas.
- Leases are used to maintain a consistent mutation order across replicas.
  - The master grants a chunk lease to one of the replicas [Primary]
  - The primary picks a serial order for all mutations to the chunk.
  - All replicas follow this order when applying mutations.
  - Thus, the global mutation order is defined by
    - the lease grant order chosen by the master, and
    - within a lease by the serial numbers assigned by the primary.

Handling Write Operations

- The client asks the master which chunk server holds the current lease
  - The master replies with the identity of the primary and the locations of the other (*secondary*) replicas
  - The client caches this data for future mutations
- The client pushes the data to all the replicas
  - Data flows from Client → Replica 1 → Replica 2 …
- Once all the replicas have acknowledged receiving the data, the client sends a write request to the primary

Handling Write Operations

- The primary assigns consecutive serial numbers to all the mutations it receives, possibly from multiple clients
  - It applies the mutation to its own local state in serial number order
- The primary forwards the write request to all secondary replicas.
  - Each secondary replica applies mutations in the same serial number order assigned by the primary
  - The secondaries all reply to the primary indicating that they have completed the operation.
- The primary replies to the client

Atomic Record Appends

- GFS provides an atomic append operation called *record append*.
  - Client pushes the data to all replicas and sends its request to the primary.
  - Primary checks to see if appending the record to the current chunk would cause the chunk to exceed the maximum size (64 MB).
    - If so, it pads the chunk to the maximum size, tells secondaries to do the same, and replies to the client indicating that the operation should be retried on the next chunk.
    - If the record fits within the maximum size, the primary appends the data to its replica, tells the secondaries to write the data at the exact offset where it has, and finally replies success to the client.
  - If a record append fails at any replica, the client retries the operation.
  - As a result, replicas of the same chunk may contain different data possibly including duplicates of the same record in whole or in part.
- GFS does not guarantee that all replicas are bytewise identical
  - It only guarantees that the data is written at least once as an atomic unit.

Chunk Placement and Replication

- When the master *creates* a chunk, it chooses where to place the initially empty replicas
  - Place new replicas on chunk servers with below-average disk space utilization
  - Limit the number of "recent" creations on each chunk server
  - Spread replicas of a chunk across racks.
- Master may need to re-replicate chunks
  - Chunk server becomes unavailable
  - Replica may be corrupted
  - Disk errors
  - Increased replication goal
- The master *rebalances* replicas periodically: it examines the current replica distribution and moves replicas for better disk space and load balancing

Garbage Collection

- Lazy Garbage Collection
  - When a file is deleted by the application, the master logs the deletion immediately just like other changes
  - The file is just renamed to a hidden name that includes the deletion timestamp.
  - During the master's regular scan of the file system namespace, it removes any such hidden items
  - Until then, the file can still be read under the new, special name and can be undeleted by renaming it back to normal.
  - In a similar regular scan of the chunk namespace, the master identifies orphaned chunks and erases the metadata for those chunks.
  - In a *HeartBeat* message regularly exchanged with the master, each chunkserver reports a subset of the chunks it has, and the master replies with the identity of all chunks that are no longer present in the master's metadata.
  - The chunkserver is free to delete its replicas of such chunks.