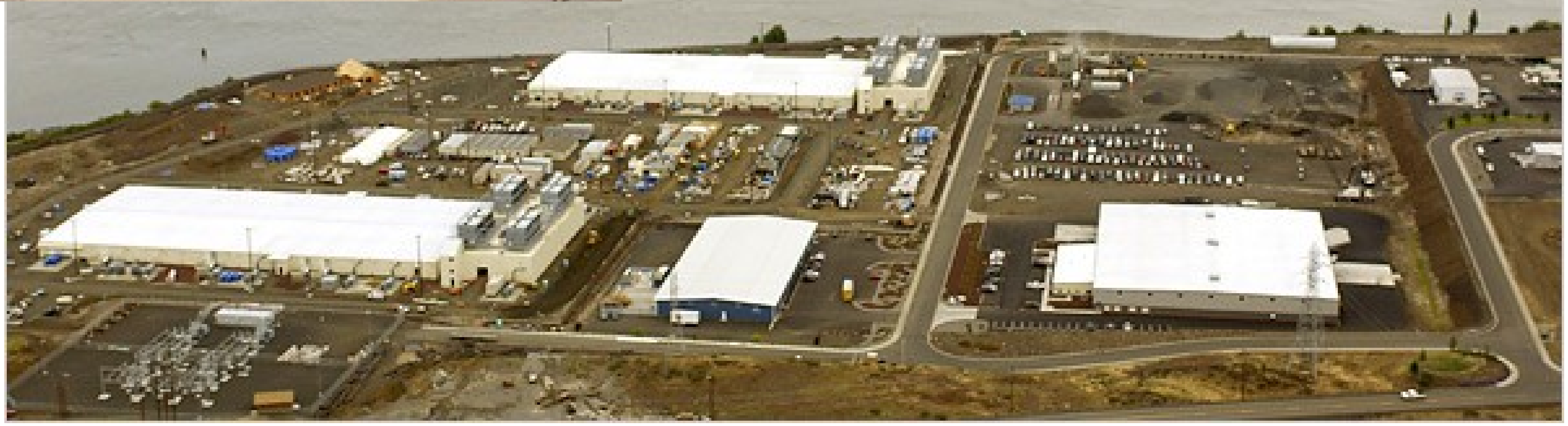
Google File System

Google Disk Farm

Early days…

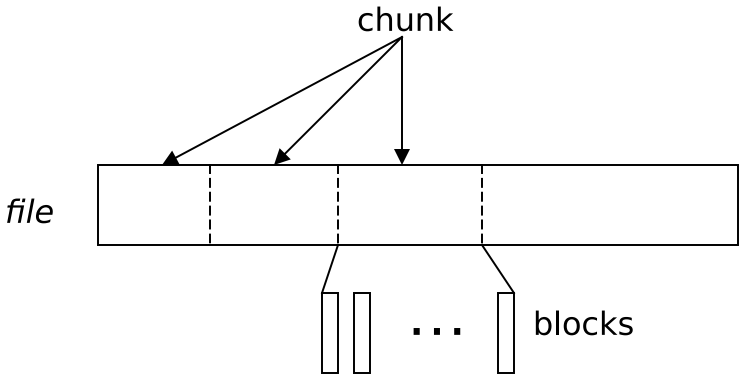
…today



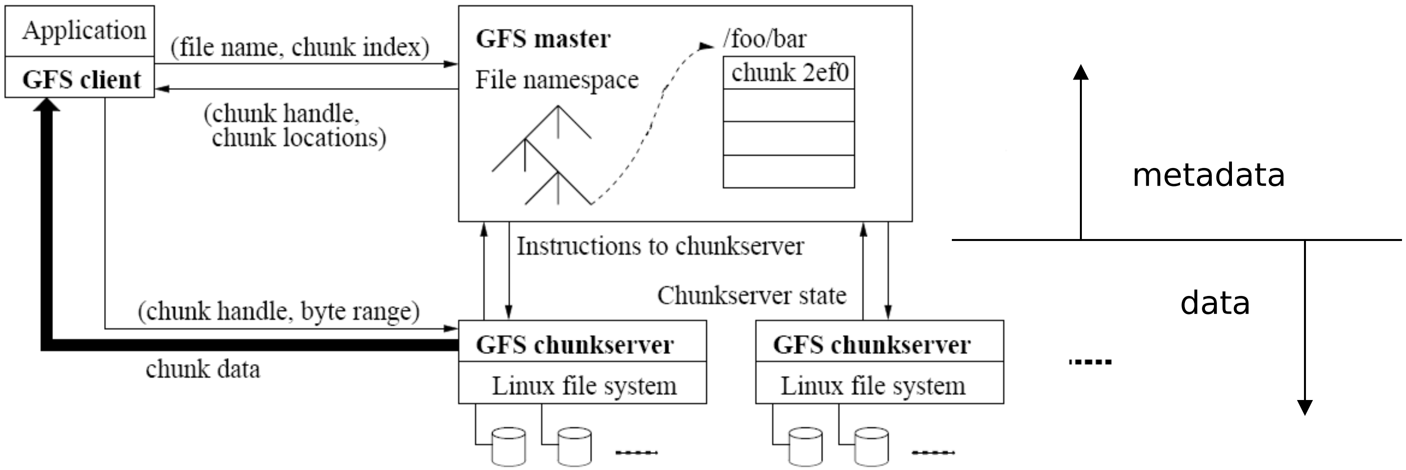
Design

- Design factors
  - □ Failures are common (built from inexpensive commodity components)
  - □ Files
    - large (multi-GB)
    - mutation principally via appending new data
    - low-overhead atomicity essential
  - □ Co-design applications and file system API
  - □ Sustained bandwidth more critical than low latency

- File structure
  - □ Divided into 64 MB chunks
  - □ Chunk identified by 64-bit handle
  - □ Chunks replicated (default 3 replicas)
  - □ Chunks divided into 64KB blocks
  - □ Each block has a 32-bit checksum



Architecture

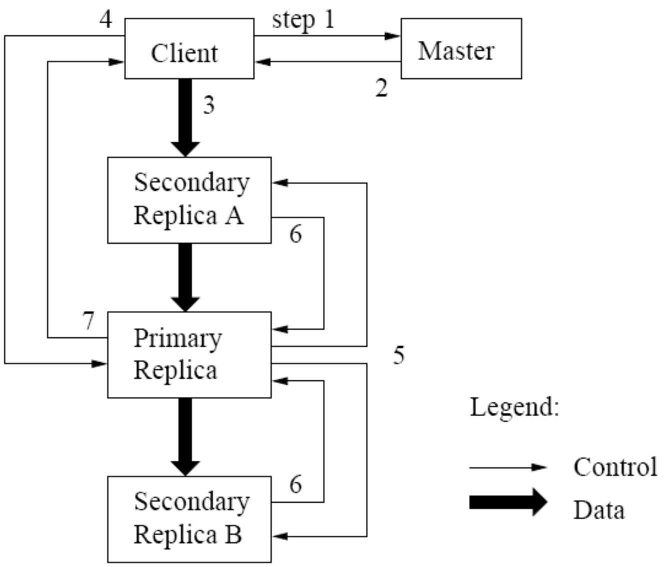


- Master
  - Manages namespace/metadata
  - Manages chunk creation, replication, placement
  - Performs snapshot operation to create duplicate of file or directory tree
  - Performs checkpointing and logging of changes to metadata

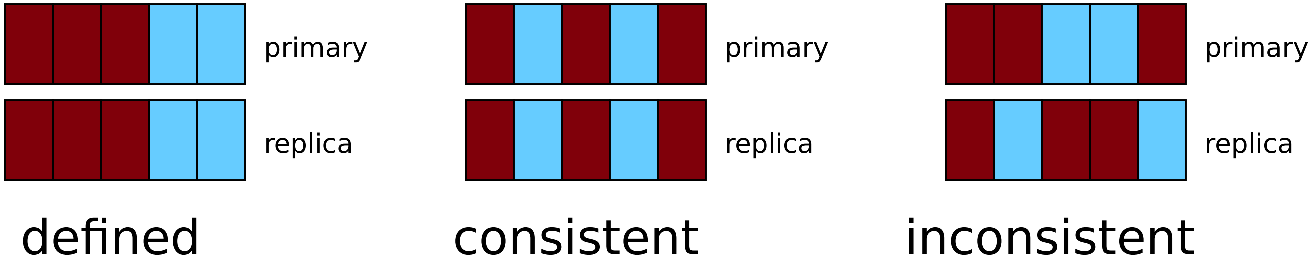
- Chunkservers
  - Stores chunk data and checksum for each block
  - On startup/failure recovery, reports chunks to master
  - Periodically reports sub-set of chunks to master (to detect no longer needed chunks)

Mutation operations

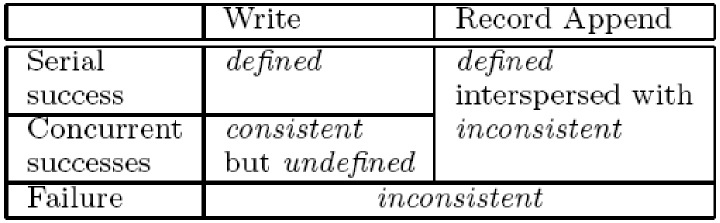
- Primary replica
  - Holds lease assigned by master (60 sec. default)
  - Assigns serial order for all mutation operations performed on replicas
- Write operation
  - 1-2: client obtains replica locations and identity of primary replica
  - 3: client pushes data to replicas (stored in LRU buffer by chunk servers holding replicas)
  - 4: client issues update request to primary
  - 5: primary forwards/performs write request
  - 6: primary receives replies from replica
  - 7: primary replies to client
- Record append operation
  - Performed atomically (one byte sequence)
  - At-least-once semantics
  - Append location chosen by GFS and returned to client
  - Extension to step 5:
    - If record fits in current chunk: write record and tell replicas the offset
    - If record exceeds chunk: pad the chunk, reply to client to use next chunk



Consistency Guarantees



- Write
  - □ Concurrent writes may be consistent but undefined
  - □ Write operations that are large or cross chunk boundaries are subdivided by client into individual writes
  - □ Concurrent writes may become interleaved
- Record append
  - □ Atomically, at-least-once semantics
  - □ Client retries failed operation
  - □ After successful retry, replicas are defined in region of append but may have intervening undefined regions
- Application safeguards
  - □ Use record append rather than write
  - □ Insert checksums in record headers to detect fragments
  - □ Insert sequence numbers to detect duplicates



	Write	Record Append
Serial success	*defined*	*defined* interspersed with *inconsistent*
Concurrent successes	*consistent* but *undefined*	
Failure	*inconsistent*	

Metadata management

Logical structure

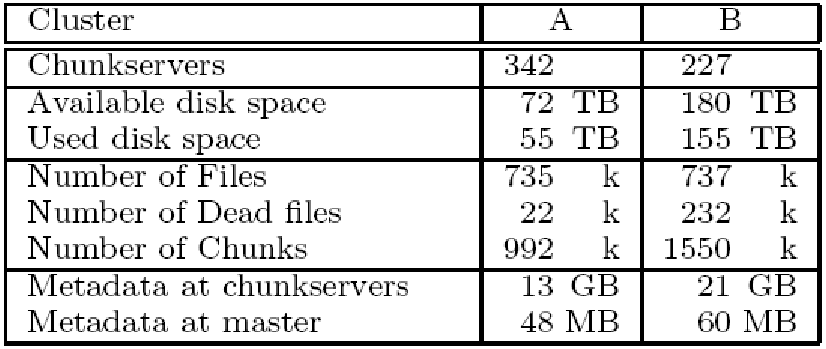
pathname	lock	chunk list
/home	read	Chunk4400488,…
/save		Chunk8ffe07783,…
/home/user/foo	write	Chunk6254ee0,…
/home/user	read	Chunk88f703,…

- Namespace
  - Logically a mapping from pathname to chunk list
  - Allows concurrent file creation in same directory
  - Read/write locks prevent conflicting operations
  - File deletion by renaming to a hidden name; removed during regular scan
- Operation log
  - Historical record of metadata changes
  - Kept on multiple remote machines
  - Checkpoint created when log exceeds threshold
  - When checkpointing, switch to new log and create checkpoint in separate thread
  - Recovery made from most recent checkpoint and subsequent log
- Snapshot
  - Revokes leases on chunks in file/directory
  - Log operation
  - Duplicate metadata (not the chunks!) for the source
  - On first client write to chunk:
    - Required for client to gain access to chunk
    - Reference count > 1 indicates a duplicated chunk
    - Create a new chunk and update chunk list for duplicate

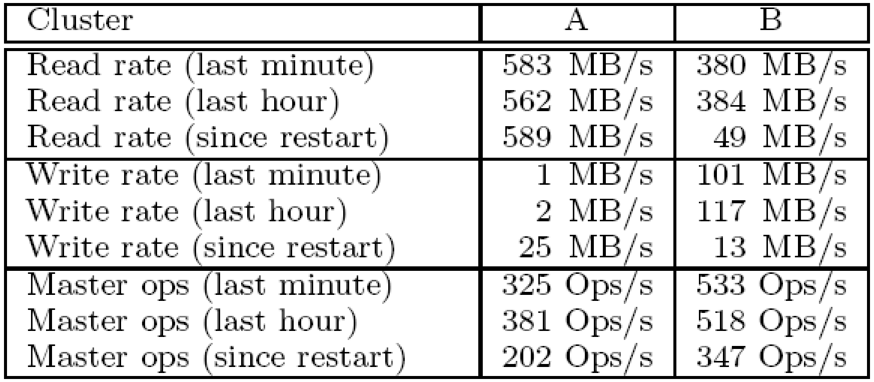
Chunk/replica management

- Placement
  - □ On chunkservers with below-average disk space utilization
  - □ Limit number of “recent” creations on a chunkserver (since access traffic will follow)
  - □ Spread replicas across racks (for reliability)
- Reclamation
  - □ Chunk become garbage when file of which they are a part is deleted
  - □ Lazy strategy (garbage college) is used since no attempt is made to reclaim chunks at time of deletion
  - □ In periodic “HeartBeat” message chunkserver reports to the master a subset of its current chunks
  - □ Master identifies which reported chunks are no longer accessible (i.e., are garbage)
  - □ Chunkserver reclaims garbage chunks
- Stale replica detection
  - □ Master assigns a version number to each chunk/replica
  - □ Version number incremented each time a lease is granted
  - □ Replicas on failed chunkservers will not have the current version number
  - □ Stale replicas removed as part of garbage collection

Performance



Cluster	A	B
Chunkservers	342	227
Available disk space	72 TB	180 TB
Used disk space	55 TB	155 TB
Number of Files	735 k	737 k
Number of Dead files	22 k	232 k
Number of Chunks	992 k	1550 k
Metadata at chunkservers	13 GB	21 GB
Metadata at master	48 MB	60 MB



Cluster	A	B
Read rate (last minute)	583 MB/s	380 MB/s
Read rate (last hour)	562 MB/s	384 MB/s
Read rate (since restart)	589 MB/s	49 MB/s
Write rate (last minute)	1 MB/s	101 MB/s
Write rate (last hour)	2 MB/s	117 MB/s
Write rate (since restart)	25 MB/s	13 MB/s
Master ops (last minute)	325 Ops/s	533 Ops/s
Master ops (last hour)	381 Ops/s	518 Ops/s
Master ops (since restart)	202 Ops/s	347 Ops/s