# GFS:
# Evolution on Fast-Forward

Kirk McKusick (BSD/BFFs) interviews Sean Quinlan (former GFS tech leader)

CACM: March 2010

# Initially

- Initial conception of Google did not include new file system
- No other choice, so GFS born
  - Monitoring, error detection, fault tolerance, auto recovery all part of file system
- Anticipated throughput requirements necessitated changing traditional assumption
  - I/O operations and block sizes
  - Scalability

# Decades later

- Store of data and applications continue to rely on GFS
- Adjustments have been made to file system along the way

# Single-master

- Unorthodox decision to base GFS on single-master design (1st decision)
  - Bandwidth bottleneck, single point of failure
- Simplified overall problem design
  - Central place for replication and garbage collection
- Faster roll out
- BigTable built later is distributed (took many years)

# Original GFS review

- Chunk – 64 MB
- Chunk server – multiple ones
- Single master – meta data
  - Sophisticated chunk placement
  - Minimize involvement in read/write
    - File, chunk namespaces
    - Mapping files to chunks
    - Location of chunk's replicas
  - Client asks master which chunk to contact for data
  - Logs maintained – record of critical metadata changes
    - Used for recovery
    - Replicated  so reliable

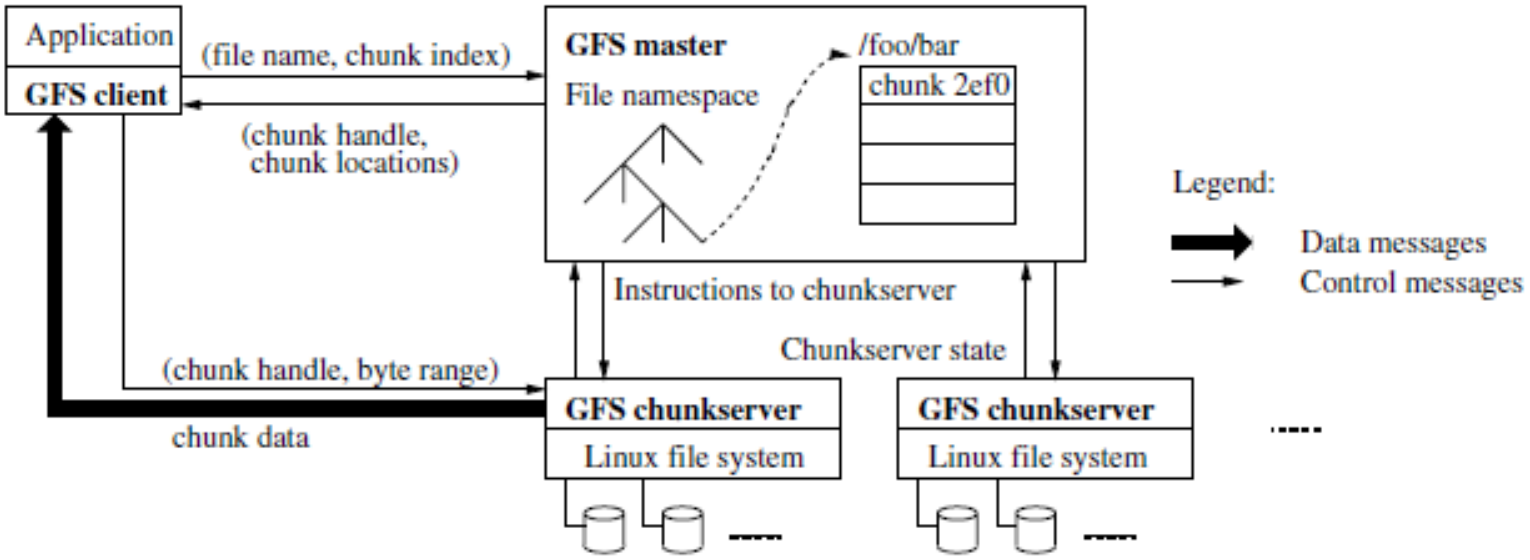


**Figure 1: GFS Architecture**

# Reality

- Initially assumed hundreds of TB, few M files
- Once size of underlying storage increased to PB, and tens PBs, problems arose
  - Operations to scan metadata for recoveries scaled linearly with volume of data
  - Amount of work of master and storage grew as well
  - Bottleneck for clients even though clients issue few metadata operations themselves
    - Open causes involvement of master

# Problems

- Master can complete only a few thousand ops per second
- MapReduce may have a thousand tasks wanting to open a number of files

# Current Implementation

- One master per cell
- Historically, goal of one cell per data center, but ended up with multi-cell approach
  - More than one cell per data center
  - Cells across network functions as related but distinct file systems

# Current Implementation

- Put multiple GFS masters on top of pool of chunkservers
  - Could be configured to have multiple GFS masters
    - Gives a pool of underlying storage
  - Application responsible for partitioning data across those different cells
  - Assume each application own master (uses one master or small set of masters)
    - Name Spaces hides all of this from application, static way to partition namespace
    - Logs Processing System - Once logs exhaust one cell, move to multiple GFS cells
    - Namespace file describes how log data is partitioned across different cells

# Current Implementation

- Put effort into tuning master performance
- Atypical at Google to tune any one particular binary
  - Just get things working reasonably well and then focus on scalability
  - Making master lighter weight – paid off
    - When scaled from 1Ks operations to 10Ks operations, master became bottleneck

# Reflections

- GFS ready for production in record time, team of 3 responsible, readied for deployment in less than a year
- Assume GFS is largest file system in operation
- Google quickly surpasses could orders magnitude of growth
- Original consumer of GFS was large crawling and indexing system
- Second wave used GFS to store large data sets
- GFS adjusted to accommodate new use cases
- Applications also developed with GFS in mind

# Additional problems

- With rapid growth, 64MB chunk size not as great
  - Large size
  - Many files need less than 64 MB (Gmail)
- File counts also a problem
  - Number of logs increases
    - Front end server would write logs
    - Front end crashes, more logs written
  - More data than had anticipated

# Additional Improvements

- File count growth a problem
  - Only a finite number of files can accommodate before master runs out of memory
- Need metadata about each file stored in memory
  - Stores file identity and chunks
    - If average file size below 64MB, ratio of number of objects on master to storage decreases
  - To deal with this
    - Combine objects into larger files, create table of contents for it
    - Put quotas on file counts and storage space
      - Limit people run into is usually file count quota

# Additional Improvements

- Working on whole new design
  - Distributed multimaster model
  - New file size of 1MB
    - Helps file count
    - Reading 10,000 10KB more seek time than 100 1MB files
    - 100M files per master, 100s masters

# BigTable

- Distributed storage system for structured data
  - Remedy for file-count – aggregate small things
  - Scales to PB across thousands of machines
  - Built on GFS, runs on GFS, designed to be consistent with GFS principles
  - While seen as application, really an infrastructure piece
    - Very failure-aware system
    - Used for crawling and indexing systems
    - Any app with lots of small data items use BigTable
    - BigTable intended for more than just dealing with file count problem

# BigTable

- Original idea to have only2 basic structures
  - SSTables – Stored String Tables (key-value pair)
  - Logs
- BigTable built on top of logs (mutable ‘stuff’) and SSTables (immutable) – data compacted
- People are free to write any sort of data they like, but majority use these 2 data structures of BigTable, SSTables
  - Provide compression and checksums

# More initial GFS limitations/ improvements

- Initial GFS design for high bandwidth (throughput) over low latency
  - Single point of failure OK for batch applications
  - Not good for video serving
  - Example
    - Write in triplicate to file. If chunkserver dies or hiccups, replicate one of the chunks – 10 seconds to a minute for recovery
    - OK if large MapReduce operations, but if Gmail, 1 minute delay not acceptable
    - Initial master failover required minutes, now down to tens of seconds

# More initial GFS limitations/ improvements

- Moved from MapReduce-based world to interactive world relying on things such as BigTable (Gmail)
- Trying to build interactive DB on top of file system designed for batch-oriented ops is challenge
- Do things such as:
  - Bigtable – transaction log is bottleneck
    - Open 2 logs at a time, write to one, if gets stuck write to the other, merge when done
  - Gmail is multihomed
    - If one instance of Gmail account stuck, moved to another data center (also helps availability)

# More initial GFS limitations/ improvements

- Compatibility issues:
  - Mismatches between drivers and drives caused corrupt data (didn't support all IDE protocol versions)
    - Rigorous end-to-end checksumming
    - But – reading slightly stale data OK
      - Could have remedied this by adding more data into master and maintain more state
    - Designers making decisions owned file system and also applications intended to run on file system

# More initial GFS limitations/ improvements

- Consistency Issues
  - Can obtain different data if read given file multiple times – Some at Google see this as a problem
    - Can miss append after open file
  - GFS required everything be written to all replicas before can continue, problems if client crashes
    - Have tightened window for eventual consistency
  - RecordAppend, multiple writers can append to a log, "loose consistency"
    - No guarantees every replica written, data could be written more than once in a chunk, in different order on different chunks, etc.
      - If a problem, new primary picks new offset
    - Not expectation of file
    - Loose consistency turned out to "be more painful than it was worth"
    - Need single writer per file, serialize writes through single process for consistency

# Initial GFS aspect that still works

- Snapshot of chunk
  - For replacing replica (recovery)
    - Revoke lock so client can't write (affects latency)
  - Also support snapshot feature of GFS – clone
    - Not used widely even though very powerful
    - Difficult to implement, but tried to do it as true snapshot (unlike many other early decisions)

# Conclusions

- GFS report card is positive 10 years later
- Can't argue with success, staying power
- Scale and application mix beyond anything imagined in late 1990s
- Challenges
  - Supporting user-facing, latency sensitive applications on top of system designed for batch-system throughput
  - BigTable helped, but not a great fit for GFS
    - Makes bottleneck limitations more apparent
- Designing new distributed master system to take full advantage of BigTable