The Google File System

Sanjay Ghemawat, Howard Gobioff,
and Shun-Tak Leung

SOSP 2003

Chris Hill
CMSC818K
Sussman
Spring 2011

(These slides modified from Alex Moshchuk, University of Washington – used during Google lecture series.)

Outline

- Filesystems Overview
- GFS (Google File System)
  - Motivations
  - Architecture
  - Algorithms
- HDFS (Hadoop File System)

Filesystems Overview

- Permanently stores data
- Usually layered on top of a lower-level physical storage medium
- Divided into logical units called “files”
  - Addressable by a *filename* (“foo.txt”)
  - Usually supports hierarchical nesting (directories)
- A file *path* = relative (or absolute) directory + file name
  - /dir1/dir2/foo.txt

Distributed Filesystems

- Support access to files on remote servers
- ***Must*** support concurrency
  - Make varying guarantees about locking, who “wins” with concurrent writes, etc...
  - Must gracefully handle dropped connections
- Can offer support for replication and local caching
- Different implementations sit in different places on complexity/feature scale

Motivation

- Google needed a good distributed file system
  - Redundant storage of massive amounts of data on **cheap** and **unreliable** computers
- Why not use an existing file system?
  - Google's problems are different from anyone else's
    - Different workload and design priorities
  - **GFS** is designed for **Google apps** and **workloads**
  - **Google apps** are designed for **GFS**

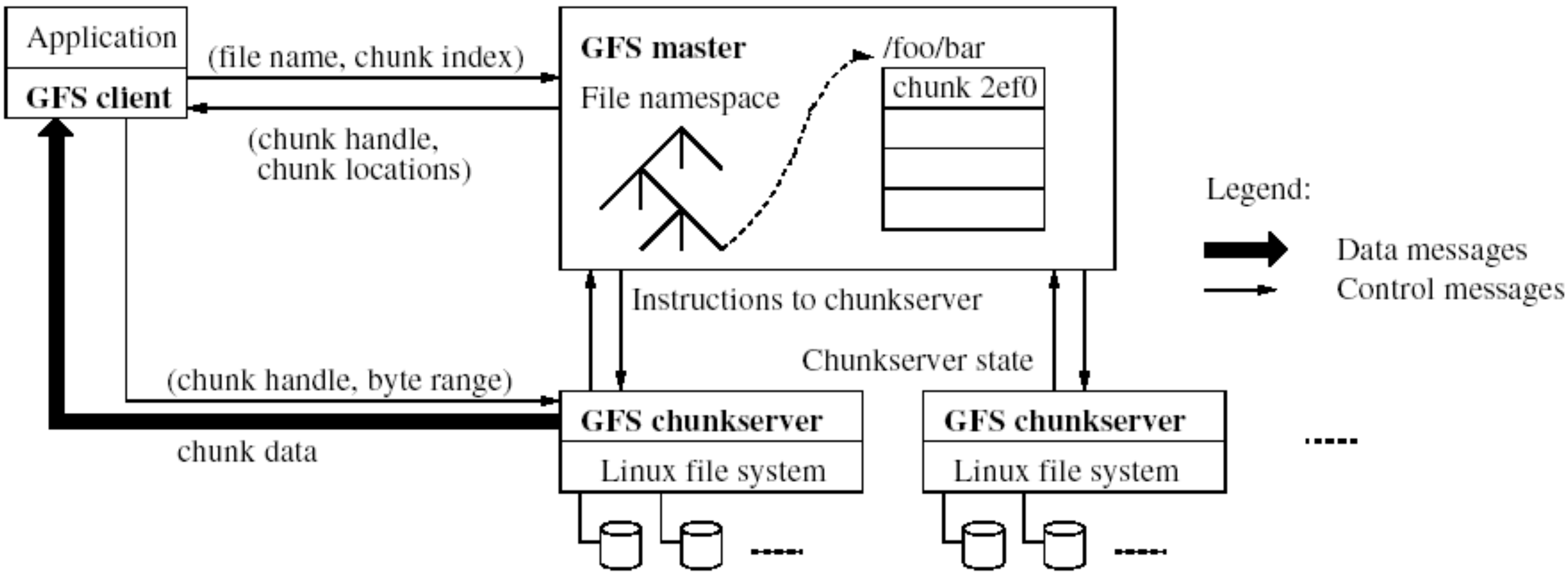
Assumptions

- **High** component failure rates
  - Inexpensive commodity components fail all the time
- "Modest" number of HUGE files
  - Just a few million
  - Each is 100MB or larger; multi-GB files typical
- Files are write-once, mostly appended to
  - Perhaps concurrently
- Large streaming reads
- High sustained throughput favored over low latency

GFS Design Decisions

- Files stored as chunks
  - Fixed size (64MB)
- Reliability through replication
  - Each chunk replicated across 3+ *chunkservers*
- **Single** master to coordinate access, keep metadata
  - Simple centralized management
- ***No*** data caching
  - Little benefit due to large data sets, streaming reads
- Familiar interface, but customize the API
  - Simplify the problem; focus on Google apps
  - Add ***snapshot*** and ***record append*** operations

GFS Architecture



…Can anyone see a potential weakness in this design?

Single master

- Problem:
  - **Single** point of failure
  - Scalability bottleneck
- GFS solutions:
  - *Shadow* masters
  - Minimize master involvement
    - **never** move data through it, use only for metadata
      - and cache metadata at clients
    - large chunk size
    - master delegates authority to primary replicas in data mutations (chunk leases)
- Simple, and good enough for Google's concerns

Metadata

- Global metadata is stored on the master
  - File and chunk namespaces
  - Mapping from files to chunks
  - Locations of each chunk's replicas
- All in memory (64 bytes / chunk)
  - Fast
  - Easily accessible

Metadata

- Master has an ***operation log*** for persistent logging of critical metadata updates
  - Persistent on local disk
  - Replicated
  - Checkpoints for faster recovery

Master's Responsibilities

- Metadata storage
- Namespace management/locking
- Periodic communication with chunkservers
  - give instructions, collect state, track cluster health
- Chunk creation, re-replication, rebalancing
  - balance space utilization and access speed
  - spread replicas across racks to reduce correlated failures
  - re-replicate data if redundancy falls below threshold
  - rebalance data to smooth out storage and request load

Master’s Responsibilities

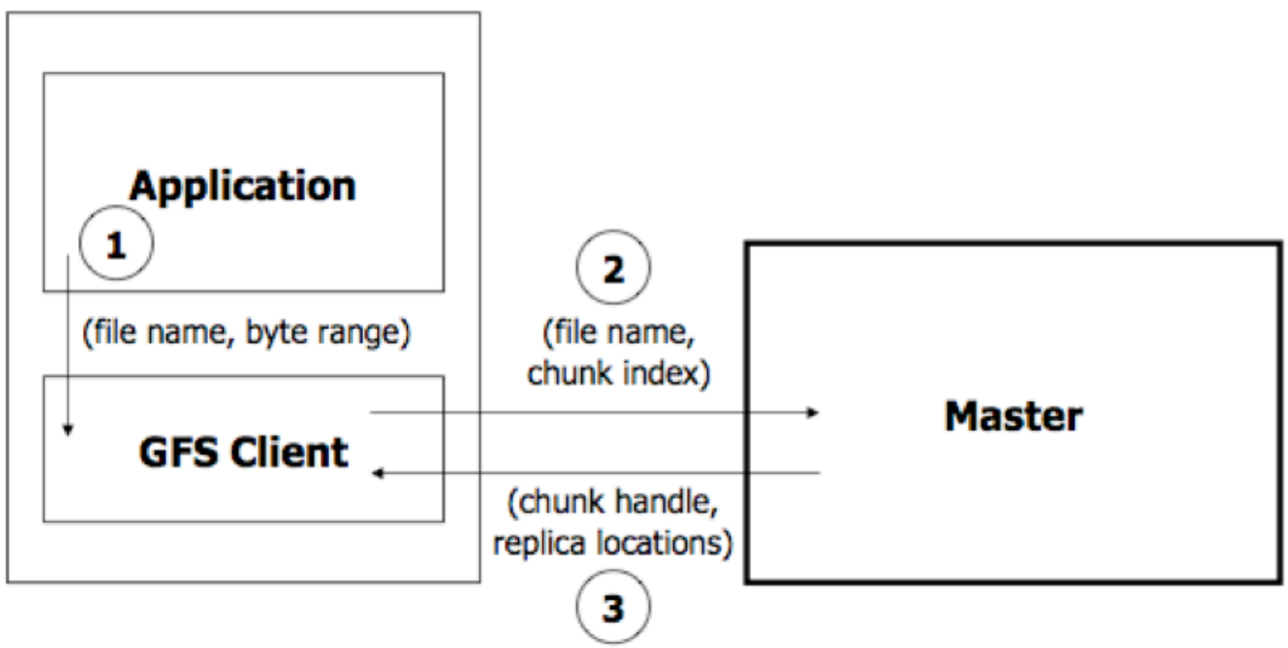
- Garbage Collection
  - simpler, more reliable than traditional file delete
  - master logs the deletion, renames the file to a hidden name
  - lazily garbage collects hidden files
- Stale replica deletion
  - detect “stale” replicas using chunk version numbers

Mutations

- Mutation = write or record append
  - Must be done for all replicas
- Goal: ***minimize*** master involvement
- Lease mechanism:
  - Master picks one replica as primary; gives it a “lease” for mutations
- Data flow decoupled from control flow

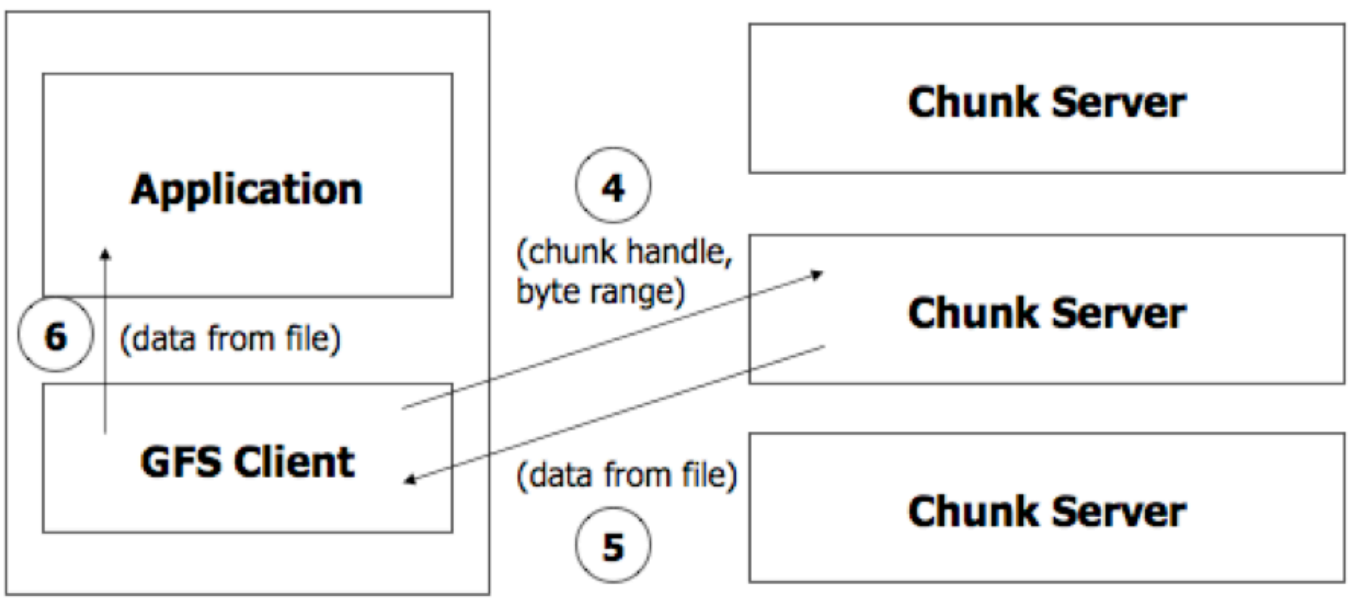
Read Algorithm

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



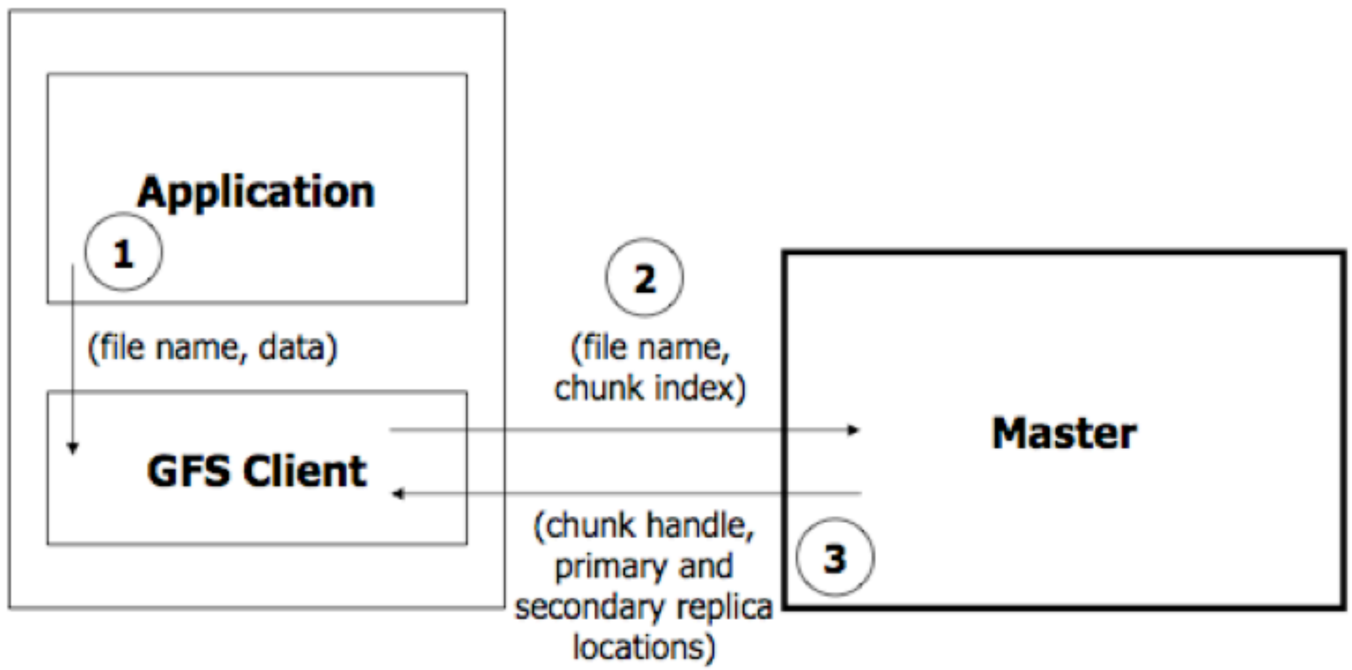
Read Algorithm

4. Client picks a location and sends the request
5. Chunkserver sends requested data to the client
6. Client forwards the data to the application



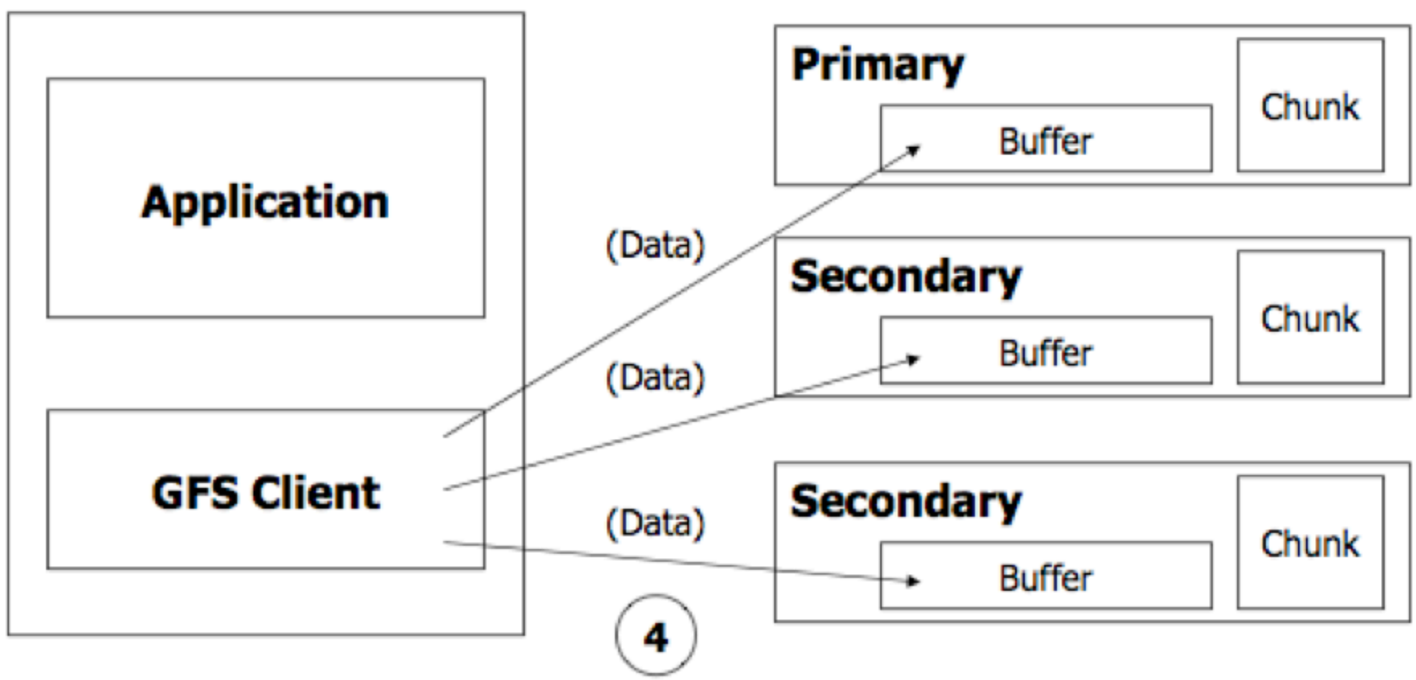
Write Algorithm

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



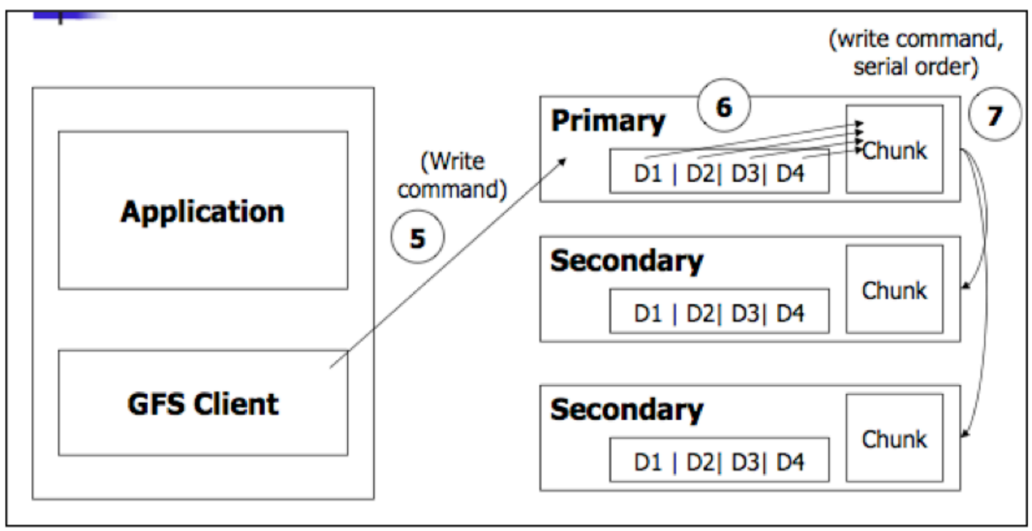
Write Algorithm

4. Client pushes write data to all locations. Data is stored in chunkserver's internal buffers



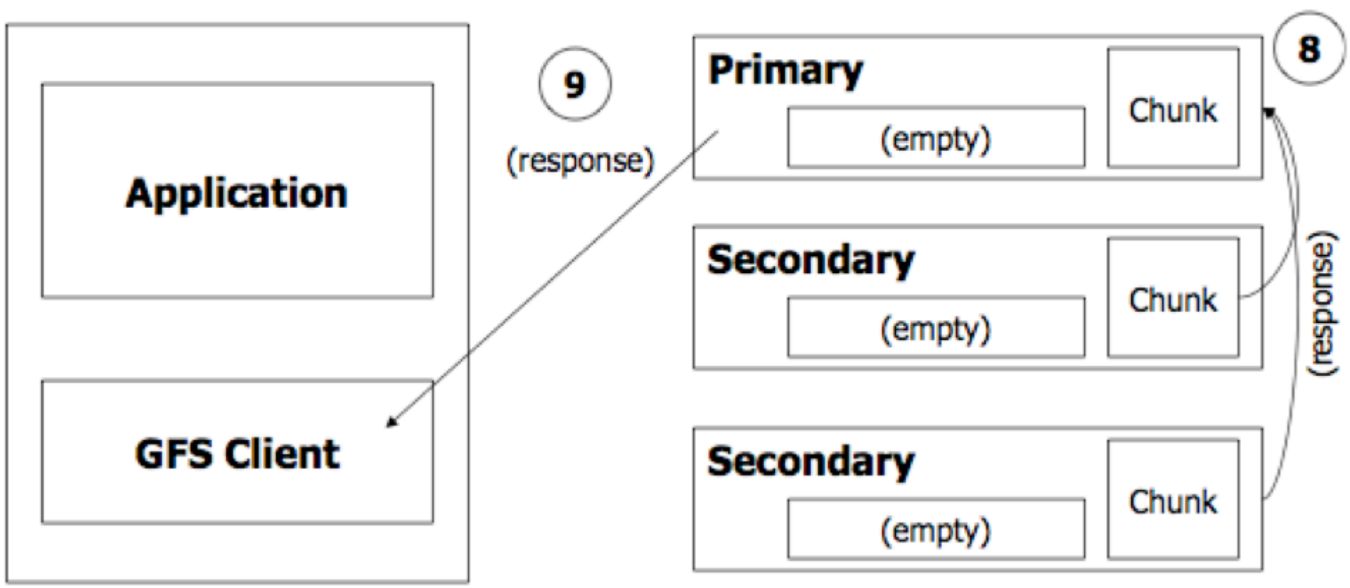
Write Algorithm

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



Write Algorithm

8. Secondaries respond back to primary
9. Primary responds back to the client



Atomic Record Append

- GFS appends it to the file atomically at least once
  - GFS ***picks*** the offset
  - Works for concurrent writers
- Used heavily by Google apps
  - e.g., for files that serve as multiple-producer/single-consumer queues
  - Merge results from multiple machines into one file

Record Append Algorithm

- Same as write, but no offset and…
  1. Client pushes write data to all locations
  2. Primary checks if record fits in specified chunk
  3. If the record does not fit:
     1. Pads the chunk
     2. Tells secondary to do the same
     3. Informs client and has the client retry
  4. If record fits, then the primary:
     1. Appends the record
     2. Tells secondaries to do the same
     3. Receives responses and responds to the client

Relaxed Consistency Model

- **Consistent** = all replicas have the same value
- **Defined** = replica reflects the mutation, consistent
- Some properties:
  - concurrent writes leave region ***consistent***, but possibly ***undefined***
  - failed writes leave the region ***inconsistent***
- Some work has moved into the applications:
  - e.g., self-validating, self-identifying records
- "Simple, efficient"
  - Google apps can live with it

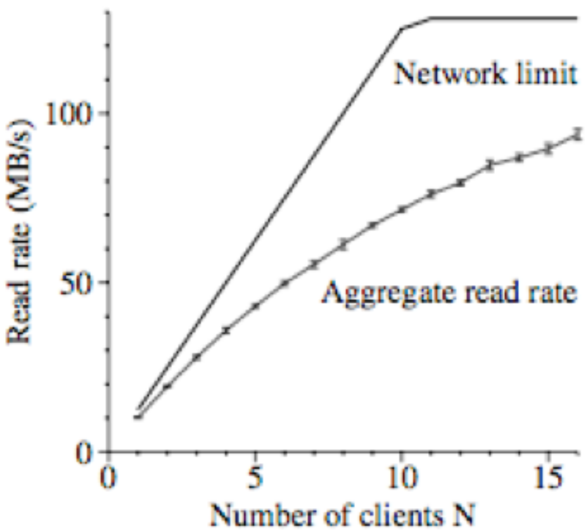
Fault Tolerance

- High availability
  - Fast recovery
    - master and chunkservers restartable in a few seconds
  - Chunk replication
    - default: 3 replicas.
  - Shadow masters
- Data integrity
  - Checksum every 64KB block in each chunk

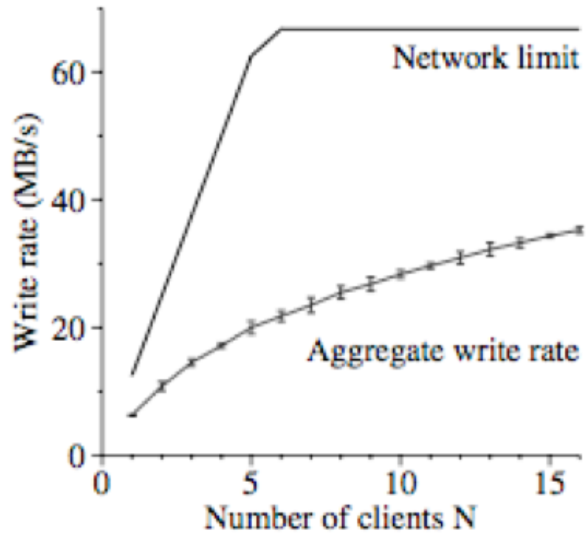
Performance Test

- Cluster setup:
  - 1 master
  - 16 chunkservers
  - 16 clients
- Server machines connected to central switch by 100 Mbps Ethernet
- Switches connected with 1 Gbps link

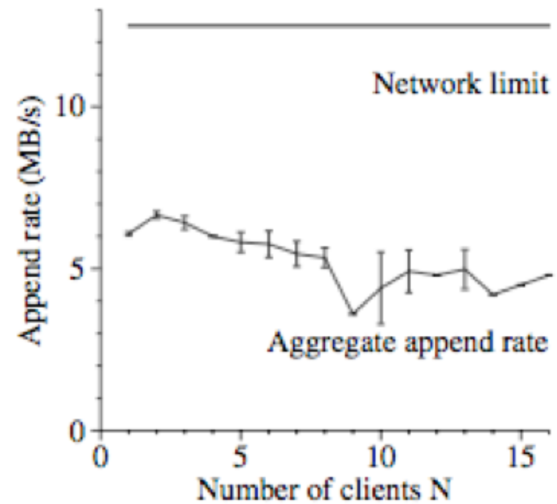
Reads



Write



Record Append



- 1 client:
  - 10 MB/s, 80% limit
- 16 clients:
  - 6 MB/s, 75% limit

- 1 client:
  - 6.3 MB/s, 50% limit
- 16 clients:
  - 35 MB/s, 50% limit
  - 2.2 MB/s per client

- 1 client:
  - 6 MB/s
- 16 clients:
  - 4.8 MB/s per client

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

Deployment in Google

- Many GFS clusters
- Hundreds/thousands of storage nodes each
- Managing petabytes of data
- GFS is under BigTable, etc.

Conclusion

- GFS demonstrates how to support large-scale processing workloads on commodity hardware
  - design **to tolerate frequent component failures**
  - optimize for **huge** files that are **mostly appended** and **read**
  - feel free to relax and extend FS interface as required
  - go for simple solutions (e.g., single master)

- GFS has met Google's storage needs, therefore good enough for them.

Hadoop File System

HDFS Design Assumptions

- Single machines tend to fail
  - Hard disk, power supply, …
- More machines = increased failure probability
- Data doesn’t fit on a single node
- Desired:
  - Commodity hardware
  - Built-in backup and failover

… Does this look familiar?

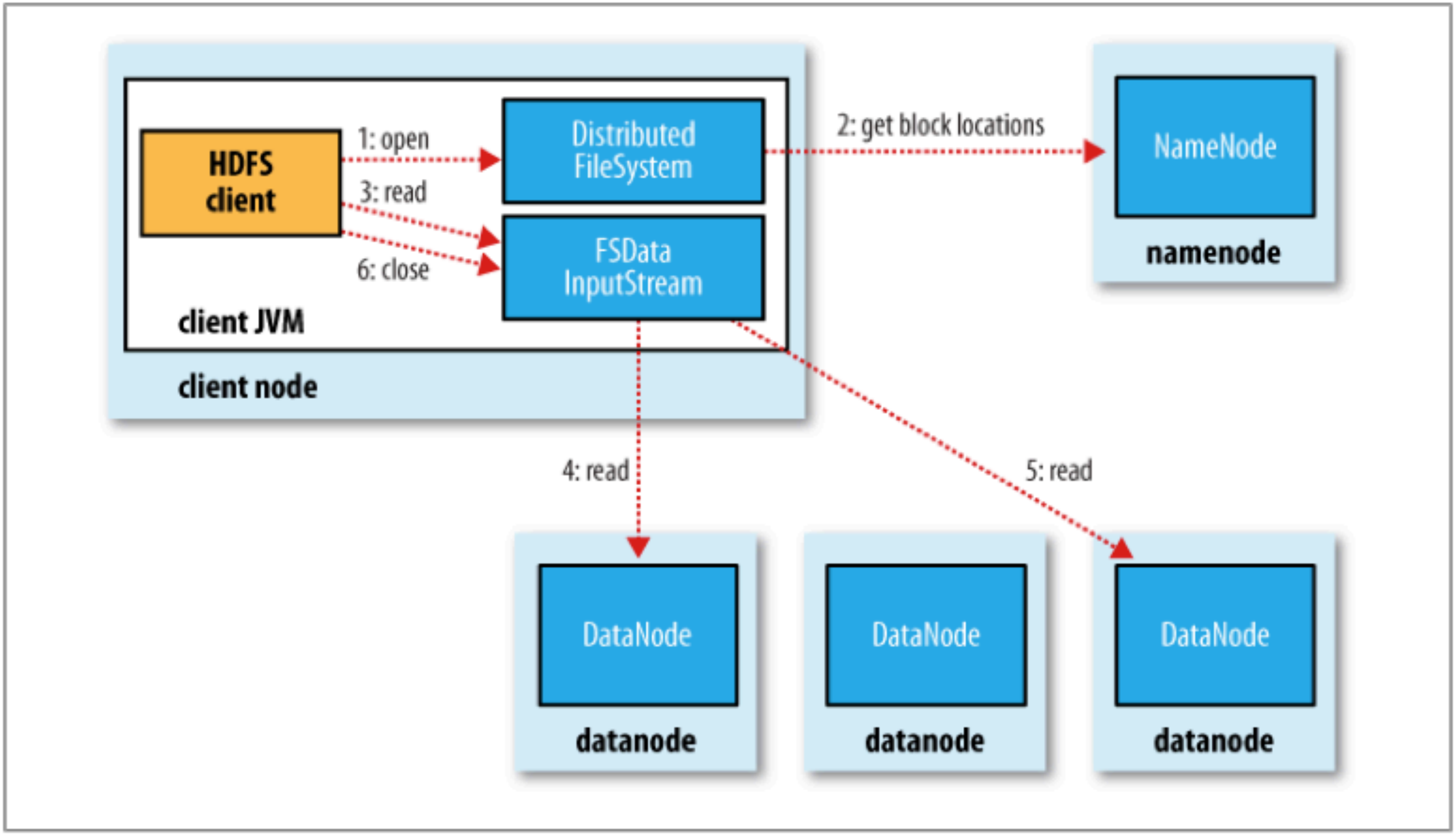
Namenode and Datanodes

- Namenode ***(Master)***
  - Metadata:
    - Where file blocks are stored (namespace image)
    - Edit ***(Operation)*** log
  - Secondary namenode ***(Shadow master)***
- Datanode ***(Chunkserver)***
  - Stores and retrieves blocks
    - …by client or namenode.
  - Reports to namenode with list of blocks they are storing

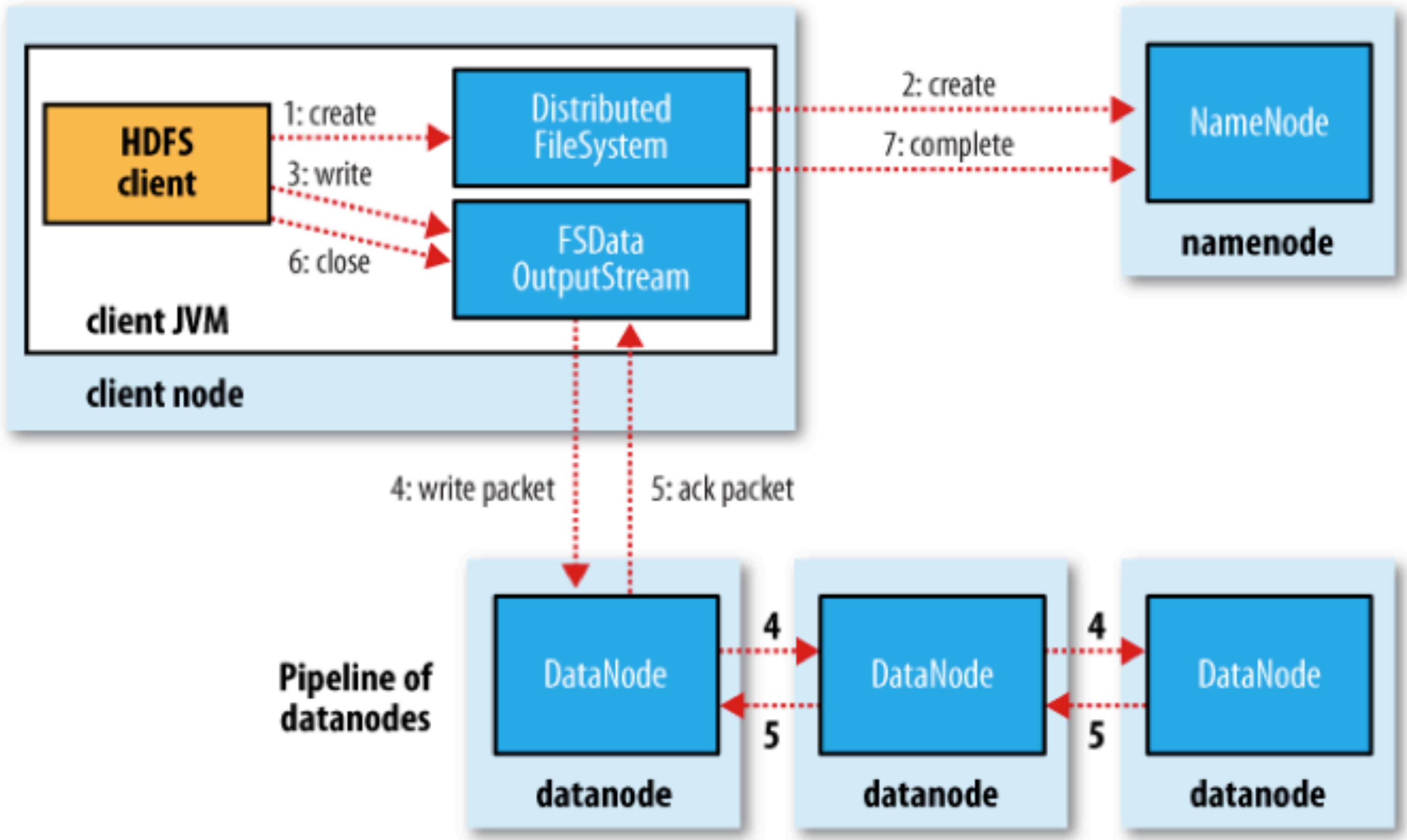
Noticeable Differences from GFS

- Only single-writers per file.
  - No ***record append*** operation.
- Open source
  - Provides *many* interfaces and libraries for different file systems.
    - S3, KFS, etc.
    - Thrift (C++, Python, …), *libhdfs* (C), FUSE

Anatomy of a File Read



Anatomy of a File Write



```java
// Print the contents of the HDFS file pathName to stdout.
public static void PrintHDFSFile(Context context, String pathName)
        throws IOException {

    // Load the HDFS library.
    Configuration conf = context.getConfiguration();
    FileSystem fs = FileSystem.get(conf);

    // Load the file input stream.
    Path hdfsPath = new Path(pathName);
    FSDataInputStream in = fs.open(hdfsPath);

    String line = null;
    while ((line = in.readUTF()) != null) {
        System.out.println(line);
    }

    in.close();
}
```

Additional Topics

- Replica placements:
  - Different node, rack, and center
- Coherency model:
  - Describes data visibility
  - Current block being written may not be visible to other readers
- Web demo

Questions?

- Additional slides taken from:
  - http://www.cs.rochester.edu/~naushad/survey/nz-google-file-system.ppt.pdf