Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

The Google File System

19th ACM Symposium on Operating Systems Principles, , October, 2003.

1. What is the motivation for this work? What are their assumptions? What do you think is most impressive about their goals?

Component failures are common. Files are huge, but not too many of them. Most files are mutated by appending new data; once written, most are only read and often only sequentially. Co-design with applications (map reduce).

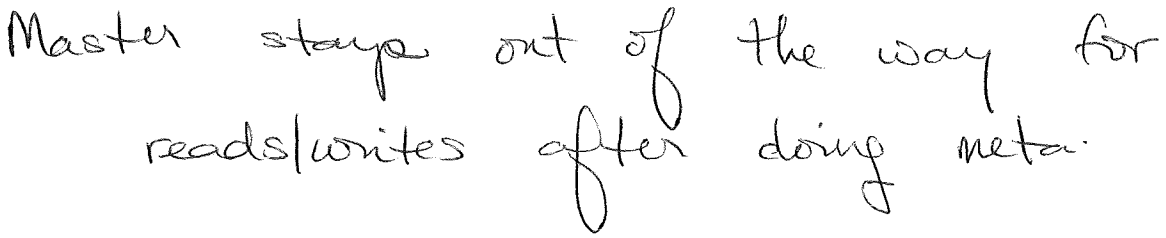
- efficient well-defined semantics for multiple clients concurrently appending
- bw more than latency

Working on 1000s of nodes in single cluster

2. What is the overall architecture of their system? Is a master the right design decision?

Single master and multiple chunkservers accessed by multiple cients.

Fig. 1



3. What are the interactions between the nodes on a read operation? Is the master likely to be a bottleneck for reads?

Allowed to cache translations from master and can batch requests.

4. What data structures are kept on the master? Which are persistent? What are not? How does the master get information ~~about~~ after a crash? What are the pros and cons of keeping all of that meta-data in main memory?

Master tracks 3 types of metadata: the file and chunk namespaces, the mapping from files to chunks, and the locations of each chunk's replaces. All kept in memory. First two are persistent by logging to local disk and replicated on remote machine. Chunk locations are soft state; asked on master startup and whenver a chunkserver joins.

Overhead in memory:
64 B for 64MB chunk required
(64B / file ignore)

300 TB of storage

→ 300 MB of meta-data?

+fast in common case
- need significant memory (but okay)

---

after server crash?
- contacted by chunkservers - informs of which it has

during server op?
- heartbeats - if no response, remove?
- if chunkserver starts up, ~~pull~~ update

5. GFS makes the design decision to not explicitly cache files on either the clients or the chunkservers. Does this seem like a good decision?

Fine. Streaming through data → low re-use

6. GFS makes the design decision to use fixed sized chunks of 64 MB? What factors argue for large chunk sizes? What factors argue for small chunk sizes? How does 64 MB interact with Map-Reduce applications? Does 64 MB seem reasonable?

Large: Reduce interactions between master; persistent TCP connection with chunkserver, reduce size of metadata stored on master (memory, 64 bytes per 64 MB chunk). Small: Small files in just one chunk, could be a hot spot. Limits size of mappers; smaller mappers are better for load-balancing.

helps fit in client cache

lazy allocation of chunks (file in local fs)

7. GFS specializes its consistency model to its application domain. To understand Table 1: What does it mean for replicas to be consistent? What does it mean to be defined? What is the difference between a write? and a record append? How do the different states occur? Why or why not are all of these states acceptable?

Consistent: All replicas have the same data. Defined:
Consistent AND will see results of mutation writes in their entirety.

Append: At least once semantics; must be able to discard duplicates; implication of failure: app must retry until successful. Concurrent writes discouraged!

successful write → defined w/o interference

concurrent → still consistent, but could be interleaved

failure → inconsistent

8. Is it ever possible for a client to read an inconsistent (i.e., stale) replica? Do you think this is acceptable?

Yes possible to read stale data, since might have the location cached;
not as big of deal if reading only from end of file since will see
premature end-of-file instead of old data.

9. What happens when a client wants to write? Why is it helpful to have a primary? Are leases appropriate here? How does the replica-update protocol achieve decent performance while ensuring that replicas are kept consistent?

§ 3.1 & Figure 2

1) Client asks master for chunkservers w/ lease + replica
2) Master responds. Client can cache.
3) client pushes data to closest replicas (sep. from control)
4) After replica ack, client write request to primary
   - If append, primary determines offset
   - Assign serial #s for ordering
5) Primary tells replicas of order
6) Replicas reply
7) Primary replies to client

Primary - keeps concurrent ops ordered

Leases: great when have failures

+ Sep data xfr + control

10. Why might the write protocol lead to inconsistent regions? undefined?

If a write fails, the data is left in an inconsistent state.
Will try op again. Undefined if interleave multiple requests across boundary (will be consistent though).

11. How is the protocol for record appends different than ordinary writes? (Why must the the primary sometimes pad the previous chunk?) Why might this protocol lead to some inconsistent entries? How do applications deal with this model?

Section 2.7.2 + 3.3

~~12.~~ How does the master organize the file namespace? What is the advantage of their approach compared to a traditional Unix directory structure?

In their workload, probably common to be creating many files in the same directory concurrenty (by multiple reducers). Important to be able to overlap.

/a/b/c	~
/a/b/e	~
/a/b/h	~

lookup table mapping full pathnames to metadata → chunk id

-prefix compression - in-memory

-2 filename creates in same dir simultaneously

- grab read locks on dir. path names ~~store file of interest~~ - can't be deleted or renamed

- grab write lock on desired file name

13. Where replicas are placed is an important factor for both reliability and performance. What is the GFS policy for placing replicas?

- Spread replicas across racks

- Read: from closest (same rack)

- Writes: more expensive, but ... worth it for reliability
  multiple copies (3)

Pick ~~below~~ are. Lik utiliz.

. limit recent creations

§ 4.4

14. What happens when a file is deleted? How is the physical space on disk actually freed? Do you think this is a better approach than having the master explicitly tell the chunkservers to delete the space?

    Good interaction with treating mapping as soft sate. Good when some creation ops succeed and others fail or when master doesn't even know about some. Can remove stale copies at this time as well.

- just rename
- hard to handle delete in d.s. w/ failure

- chunkserver: tells master of chunks
  - master replies w/ those not part of file
  - chunkserver can delete

15. What is the role of chunk version numbers in the protocol?

- update on each new lease
  - keep w/ each chunk
  - make sure have most recent version

16. As discussed so far, what is the single point of failure in the system? How do they improve availability in GFS?

- Master
- Replicated w/ shadow master (read-only)
- Quick restart

17. How do they address the concern of data integrity? Given that they have multiple replicas, why don't they just compare the data across replicas and vote? Is their approach ever inefficient?

Voting very expensive; also Failures can lead to inconsistencies that the application can handle.

64KB blocks - checksummed

inefficient if overwrite data in-place

scrubbing in bg.

18. Conclusions?

Contributions: Handling node failures so well (making location of chunks soft state and using checksums for all data), pushing some complexity into map reduce framework (tuning to application semantics with appends), simplifying system to use a single master that can handle all metadata in memory.

Developed FS that really is in use, scale is impressive. Great to see the infrastructure that is really needed \ to get something useful working; not just developing concepts for what is theoretically interesting or to push some idea to the extreme. Negative of paper (but not of system) is that they don't do a great job of saying what the new contributions are or pulling out the conceptual ideas.

WIthin Google, GFS shows its age: new applications are not all like Map-Reduce, but expected to support them all.

Production fs

Centralized master works

Separate data + control plane

Easier when know app.
+ don't need to be completely general