Based on material by

Alex Moshchuk, University of Washington

GOOGLE FILE SYSTEM (GFS)

Motivation

- Massive distributed data store
  - Redundant storage
  - **Massive** amounts of data
  - Cheap and unreliable computers
- Not general purpose
  - Data consistency checking done by application

Assumptions

- High component failure rates
- "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*

32

32

GFS Design Decisions

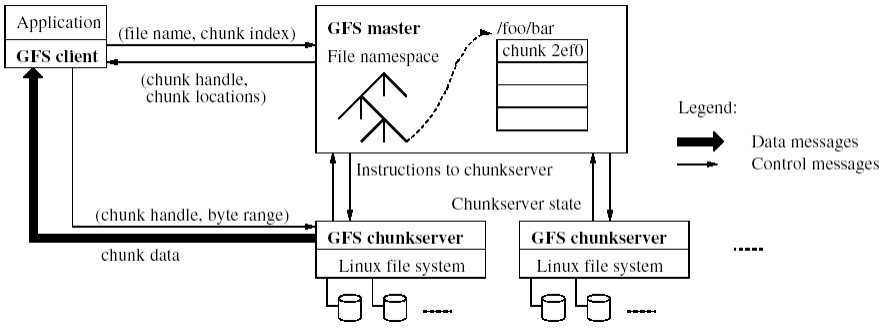
- Files stored as **chunks**
  - Fixed size (64MB)
- Reliability through **replication**
  - Each chunk replicated across 3+ chunkservers
- Single **master** to coordinate, keep metadata
- No data caching
- Familiar interface, but customize the API
  - Add **snapshot** and **record append** operations

33

33

GFS Architecture

- Single master
- Multiple chunkservers



…Can anyone see a potential weakness in this design?

Single master

- This is a:
  - Single point of failure
  - Scalability bottleneck
- GFS solutions:
  - *Shadow masters*
  - *Minimize master involvement*
    - Use only for metadata
    - large chunk size
    - Delegate authority to primary replicas for data mutations (**chunk leases**)

Metadata (1/2)

- 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 (2/2)

- **Operation log** of critical metadata updates
  - Persistent on master local disk
  - Replicated to backup master servers
  - Checkpoints for faster recovery

GFS: MUTATIONS

38

38

Mutations

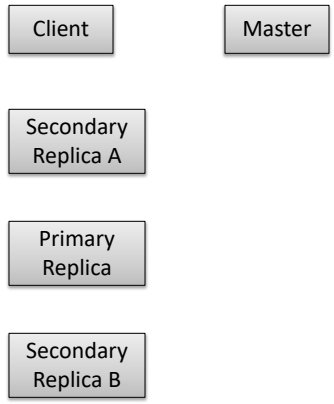
- Mutation = write or append
  - must be done for all replicas
- Goal: minimize master involvement

39

39

Mutations

- **Lease** mechanism:
  - Primary replica

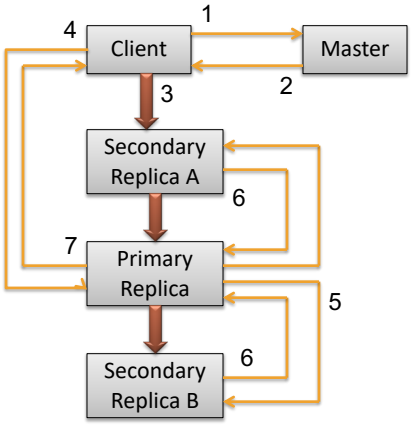


40

40

Mutations

- **Data flow** decoupled from **control flow**:
  - Master: meta-data only
  - Primary replica arranges data transfers
  - Data path has no splitting



41

41

Atomic record append

- Client specifies data
- 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., Multiple-producer/single-consumer queues

42

42

GFS: DATA CONSISTENCY

43

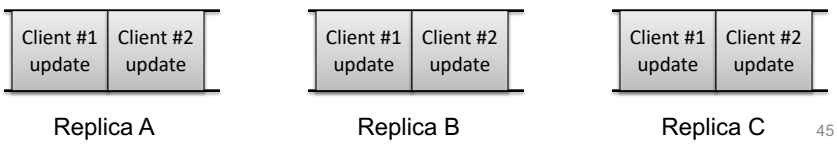
43

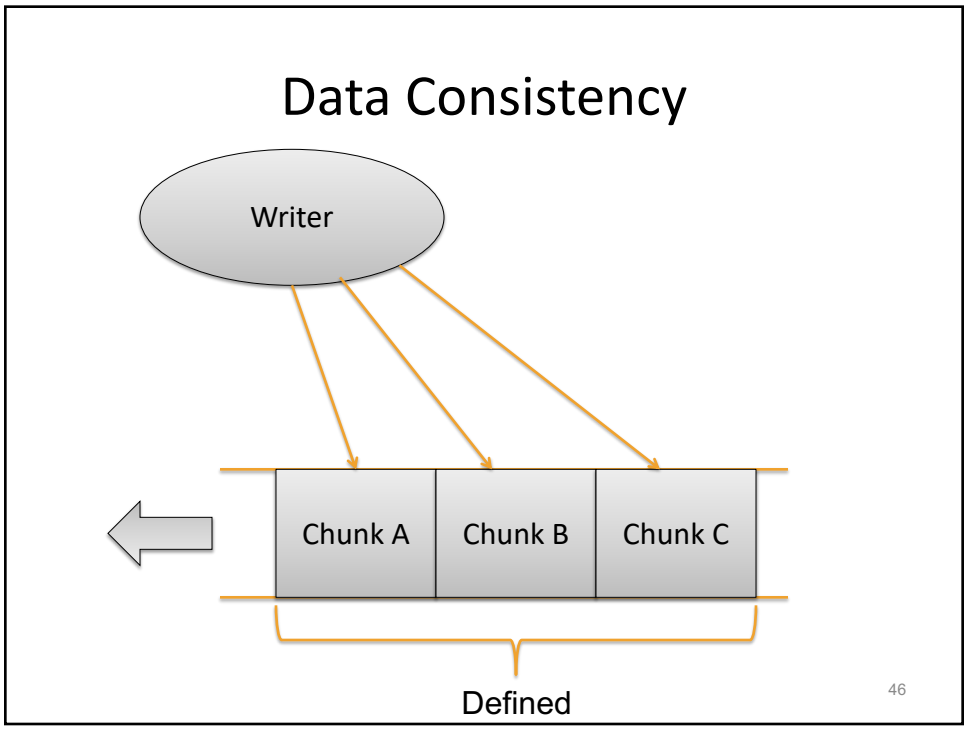
Metadata Consistency Model

- Changes to namespace (i.e., metadata) are atomic
  - Done by single master server!
  - Log defines global order

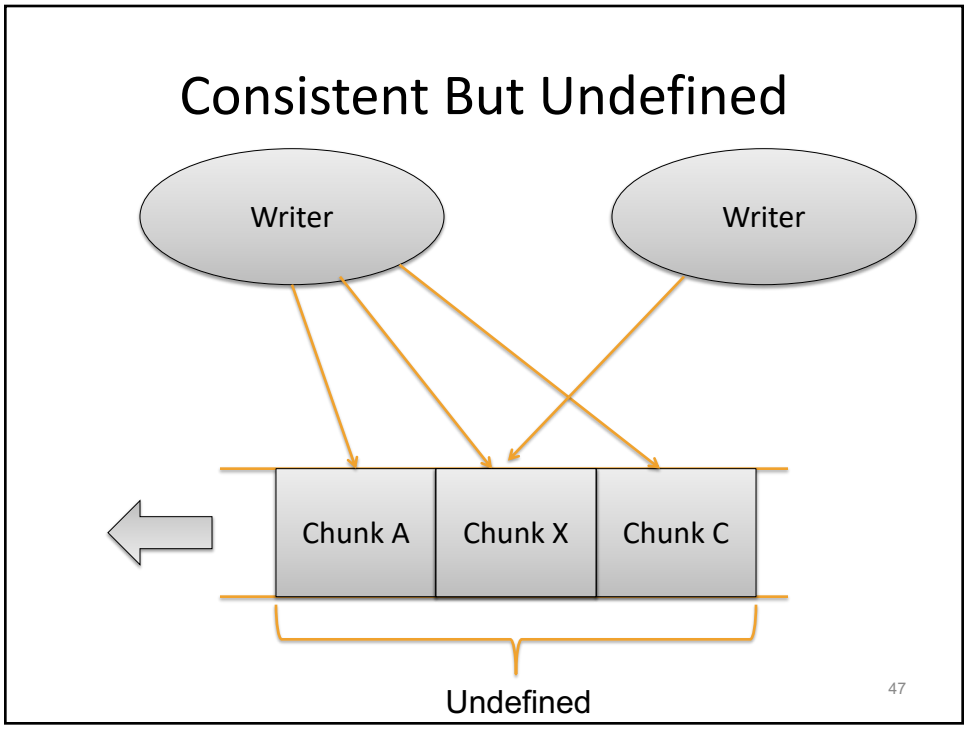
Data Consistency Model

- "Consistent" = all replicas have the same value
- Changes to data are **ordered** by a primary
  - All replicas will be "consistent"
  - But concurrent writes may be interleaved
    - Interleaving of compound updates




Data Consistency
Writer
Chunk A
Chunk B
Chunk C
Defined
46

46


Consistent But Undefined
Writer
Writer
Chunk A
Chunk X
Chunk C
Undefined
47

47

Consistent and Defined

- "Consistent" = all replicas have the same value
- "Defined" = consistent & replica reflects the mutation

- Some properties:
  - Concurrent writes leave region **consistent**
  - Concurrent writes may leave region **undefined**
    - Same corrupted write at all replicas
  - Failed writes leave the region **inconsistent**

48

48

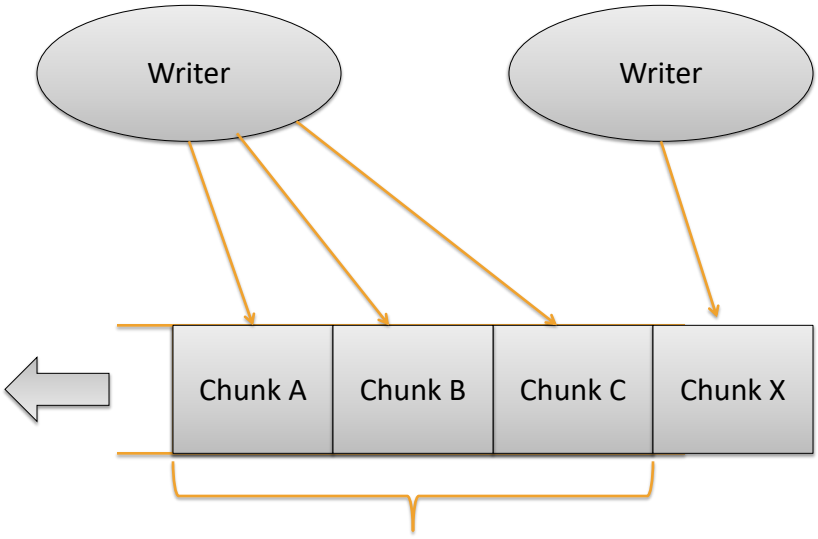
Data Consistency Model

	Write
serial success	defined
concurrent success	consistent but undefined
failure	inconsistent

49

49

Record Append



50

50

Consistent and Defined

- "Consistent" = all replicas have the same value
- "Defined" = consistent & replica reflects the mutation

- Record append completes *at least* once
  - Offset of append chosen by primary
  - *Applications* must cope with possible duplicates

51

51

Data Consistency Model

	Write	Record Append
serial success	defined	defined interspersed with inconsistent
concurrent success	consistent but undefined	
failure	inconsistent	

52

52

Implications

- Some work has moved into the applications:
  - e.g., self-validating, self-identifying records
- Namespace updates atomic and serializable
  - Single master server

53

53

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

Conclusion

- How to support large-scale processing workloads on commodity hardware
  - design for frequent component failures
  - optimize for huge files that are mostly appended and read
  - go for simple solutions (e.g., single master)