Problem Statement Tackled:

1. Component Failure

how solved?:
- failure detection
- fault tolerance
- self diagnose and repairing

2. Lack of optimized systmes for large data files

how solved?:
- optimizing for large files using chunking
- we will not be thinking a lot about small files – because they don't have that many access is what we believe
- if important files like exectuables then store it using a higher replication factor

3. Appending is prevalent, then why not a better approach for it as compared to reading LARGE complete file sequentially and then appending → - append and not overwrite is the main USECASE

How solved?:
- identified the access pattern
- random writes are NOT Optimized
- Also MULTIPLE CLINETS should be able to do the appending → this is called **PRESERVING ATOMICITY** → therefore syncing the race conditions → this is not something new but doing it in multiple client system WHEN WORKING WITH CHUNKS of data – is the main CHALLENGE

Assumptions
- LATENCY is not a concern → bandwidth is.

Key Design Principles

1. Chose to ditch POSIX and made a POSIX-like architecture with 2 additional operations: Snapshot and Atomic Append → RELAXED CONSISTENCY

2. Chunking advantages:
    a. Parallelism – multiple clinets can work on diff chunks
    b. Load balancing easy – as there might be some hot chunks
    c. Fault tolerance – done ALONG WITH replication
    d. Easy Storage allocation – since can easily find 64mb continuos space
    e. Reduced network overhead – since any operations would take less time as copmred to working on a 1 tb file as compmared to a 64mb chunk or multiple small chunks like this (it would never be even close to 1tb)

3. Working and

4. Hight Availability

5. Data integrity

6. Zero caching

7. Checksums are used to detect corruptions → how it works

8. Handling hot spots – increase the replication factor → something which is done to executable files

Architecture & Components

------ BLOCKS -----

1. Single Master Server

i. Stores metadata (names, file size, acl, chunk servers present, WHICH chunk is on which chunk server)
   a. Stores this in MEMORY
   b. What are we storing:
      i. File to chunk mapping
      ii. Location of each chunk's replica

   A decision was made – though both of these are stored in memory – we also store the file to chunk mapping on a DISK as a log file called OPERATION LOG

   OPERATION LOG
   - Any file change is first WRITTEN to the log file and then to in-memrory → This makes the recovery better and more robust
   - EVEN THE OPERATION LOG IS REPLICATED across machines → these can be chunk servers or dedicated machines – called Log Servers
   - Updation takes place periodically
   - Master takes SNAPSHOT/CHECKPOINT of the current in-memory status of the log and then FLUSHES it to the replicas on other log servers
   - This snapshot is a B tree → this choice was made because it is easier to rempa the structure on the memory/RAM (I don't know how)
     - How to verify the Integrity of the FLUSH:
       - This needs to be done because while flushing (basically writing) there might be a write failure.
       - THEREFORE the validation is done by - checksum CRC that validates the checkpoint that just got updated on log servers
   - The checkpoint is created through an async thread to not block incoming mutations.

   Therefore of memory goes out then the master atleast know what files and chunks to expect → "where are they?" it will be answered automatically in the periodic heartbeat messages

   c. Memory calculation → 64mb chunk requires 64bytes of metadata → therefore 1gb of metadata can hold info about 1petabyte

   d. How are we saving the 64bytes unique names – and then the complete path length → prefix compression can be done AT YOUR END

ii. Monitors health of chunk servers → done using heartbeat message – if no response then mark dead → then It also balances chunks
   a. In heartbeat message the chunk server also returns the chunk info that it current holds
iii. Maintains ACL (Access Control List) → which client can access which data
iv. Responsible for uniquely naming the chunks

2. Chunk Servers

i. These are the machines where the chunks are stored
ii. These chunks are not even unique as they have REPLICAS – GFS proposed 3 – across other chunk servers
iii.

3. Application & the GFS CLient in it

---- Algos ------

1. Chunk SIze / Chunking Algo & Replication across chunk servers

- Distributed such that it MAXIMIZES data reliability
- MAXIMIZE network bandwidth

Therefore chunks are spread **across racks** **(basically – on separate physical servers for more advantages)**

- A chunk is replicated when it is created **OR/AND** when one of the chunk servers go down and therefore we need to again copy/replicate the chunk into some other chunk server
- Chunk re-balancing is constanly done by master - so that we efficiently user servers/resources to access hot and cold chunks.

2. Atomic Append

3. Snapshot

------ Complete Working --------

1. Complete Working of a GFS client accessing a data chunk for read

i. Contact Master to get metadata – reduces the load on master
ii. Then based on that metadata it contact the respective chunk server and then get the required data

READ REQUEST

2. Complete Working of a GFS client accessing a data chunk for write → Leasing and Mutation Process

i. We need to have a primary replica for each chunk

UPDATING PROCESS

- Write req received by a primary replica
- All the incoming writes are not directly WRITTEN on the disk but are maintained in a LRU buffer
- The writes can be from multiple clients for the same chunk of data
- Every incoming write is assigned a serial number – eg: 1, 2, 3, 4, ... etc.
- Once the writes are done coming in → the SERIAL NUMBER ORDER is FINALIZED BY THE PRIMARY REPLICA (chunk server) – which is basically the order in which the WRITES are to be performed on the data in the disk
- This SERIAL NUMBER ORDER is then communicated by the primary replica to secondary replicas
- And then the secondary replicas can WRITE the changes that they received in the order given by Primary replica

ii. But how do we decide that which chunk is the primary replica? → by LEASE MANAGEMENT
   a. Lease assigned by master to one of the chunk servers – this data is also stored in master
   b. This lease has a TTL, therefore it expires
   c. After expiry the current primary replica can either renew this lease or it can be given to another chunk server
   d. All of this is a part of the heartbeat message.

WRITE REQUEST:

-- to be completed – in very detail

Related Work

Conclusion/Summary

Questions: during write – after the primary replica is done with mitigating the incoming writes in LRU buffer – what happens first: the ACTUAL writing on disk of primary replica **OR** the ACTUAL writing on disk of secondary replicas