FLAT DATACENTER STORAGE

CHANDNI MODI (FN8692)
OUTLINE

- Flat datacenter storage
- Deterministic data placement in fds
- Metadata properties of fds
- Per-blob metadata in fds
- Dynamic Work Allocation in fds
- Replication in fds
- Failure recovery in fds
- Failure recovery step by step
- Replicated data layout
- Failure domain
- conclusion
FLAT DATACENTER STORAGE

FDS system
DETERMINISTIC DATA PLACEMENT

- In parallel storage systems other than FDS
  - how does writer know where to send data and how does a reader find data?
    - System’s metadata server stores the location of the data blocks.
    - Writers contact metadata server to write a new block, metadata server picks the data server and stores the decision and return to writer.
    - Reader contact metadata server to find which servers store the extent to be read.
  - Adv :- allows max flexibility of data placement & visibility into system’s state.
  - Dis Adv :- metadata server can become very busy because critical path for all reads & writes and can also become central point of failure, .
In FDS

- Metadata server’s role during normal operations is simple and limited.
- When the system is initialized, tractservers locally store their position in the TLT. This means the metadata server does not need to store durable state. So it will simplify its implementation.
- So, in case of a metadata server failure, the TLT is reconstructed by collecting the table assignments from each tractserver. So we will not lose all our data in case of metadata server failure.
- It collects a list of system’s active tract server and distribute it to TLT with k-replicatation, each entry has k tractservers.
- To read or write tract number i from a blob with GUID g,
  
  Eq:- \( \text{Tract\_Locator} = (\text{Hash}(g) + i) \mod \text{TLT\_Length} \) (uses eq to replace big size table)

Q. “FDS uses a metadata server, but its role during normal operations is simple and limited:...” What are drawbacks of using a centralized metadata server? How does FDS address the issue?
Q. How does FDS locate the trackserver that stores a particular tract of a given blob? Why does FDS first identify a tract locator (an index to an entry of tract locator table) and then in the entry to find the trackserver, rather than directly identifying a trackserver using a hash function without having such a table?

- If we do not have such table then where we will put replicas of our data?
- So, in case where we do not have such table, we have just one(original) copy of the data and if something wrong happened to that data or disk then we cannot able to recover that lost data.
Q. “To be clear, the TLT does not contain complete information about the location of individual tracts in the system.” and in the GFS paper “The master maintains less than 64 bytes of metadata for each 64 MB chunk.” Compare the TLT table with GFS’s use of a full chunk-chunkserver mapping table in the context of efficiency, scalability, and flexibility.

- **In GFS**, chunk is large. So, once you ask some data location from master second time you do not need to ask location for other data in this system. This makes data excess very flexible. But single master in a system can still become potential bottleneck.

- **In FDS**, it uses a metadata server, but its role during normal operations is simple and limited i.e. it collects a list of the system’s active tractservers and distribute it to clients and this list is called tract locator table, or TLT. And when the system is initialized, tractservers locally store their position in the TLT. In case of a metadata server failure, the TLT is reconstructed by collecting the table assignments from each tractservers. So here scalability and efficiency is very good.
METADATA PROPERTIES OF FDS

- Metadata server is in critical path only when a client process starts. So tract size can be kept arbitrarily small.
- TLT can be cached since it changes only on cluster configuration (not each read & write), eliminate all traffic to metadata server in system under normal conditions.
- The metadata server stores metadata only about the hardware configuration, not about blobs. Because blob traffic is low, its implementation is simple and lightweight.
- Since the TLT contains random permutations of the list of tractservers, sequential reads and writes by independent clients are highly likely to utilize all tractservers uniformly and are unlikely to organize into synchronized convoys.
PER-BLOB METADATA

- Each blob has metadata such as its length and it is stored it in each blob’s special metadata tract (“tract −1”). Clients find a blob’s metadata on a tractserver using the same TLT used to find regular data.

- When a blob is created, the tractserver responsible for its metadata tract and creates that tract on disk and initializes the blob’s size to 0. When a blob is deleted, that tractserver deletes the metadata.

- Newly created blobs have a length of 0 tracts. Applications must extend a blob before writing past the end of it. The extend operation is atomic and is safe to execute concurrently with other clients, and returns the new size of the blob as a result of the client’s call.

- A separate API tells the client the blob’s current size. Extend operations for a blob are sent to the tractserver that owns that blob’s metadata tract. The tractserver serializes it, atomically updates the metadata, and returns the new size to each caller. If all writers follow this pattern, the extend operation provides a range of tracts and the caller may write without risk of conflict.
DYNAMIC WORK ALLOCATION

• In any system if a node falls behind, the only options for recovery is restarting its computation elsewhere. So here straggler period can represent a great loss in efficiency if most resources are idle while waiting for a slow task to complete.

• In FDS, since storage and compute are no longer colocated, the assignment of work to worker can be done dynamically, at fine granularity, during task execution. The best practice for FDS applications is to centrally give small units of work to each worker as it nears completion of its previous unit. This self-clocking system ensures that the maximum dispersion in completion times across the cluster is only the time required for the slowest worker to complete a single unit.

• Q:- “The best practice for FDS applications is to centrally (or, at large scale, hierarchically) give small units of work to each worker as it nears completion of its previous unit. This self-clocking system ensures that the maximum dispersion in completion times across the cluster is only the time required for the slowest worker to complete a single unit. Such a scheme is not practical in systems where the assignment of work to workers is fixed in advance by the requirement that data be resident at a particular worker before the job begins.” Please explain this statement.

✓ This enables FDS to mitigate stragglers a significant bottleneck in large systems because a task is not complete until its slowest worker is complete.
REPLICATION IN FDS

• To improve durability and availability, FDS supports higher levels of replication. When a disk fails, redundant copies of the lost data are used to restore the data to full replication.

❖ What happen to replicas when application writes and reads?

✓ When an application writes a tract, the client library finds the appropriate row of the TLT and sends the write to every tractserver it contains. Applications are notified that their writes have completed only after the client library receives write acknowledgments from all replicas. Reads select a single tractserver at random.

✓ Applications are notified that their writes have completed only after the client library receives write acknowledgments from all replicas.

• Replication also requires changes to CreateBlob, ExtendBlobSize, and DeleteBlob.

• When a tractserver receives one of these operations, it executes a two-phase commit with the other replicas. The primary replica does not commit the change until all other replicas have completed successfully.

• FDS also supports per-blob variable replication.
FAILURE RECOVERY IN FDS

• FDS is full bisection bandwidth networks so it can perform failure recovery dramatically faster than many other systems.
• In an n-disk cluster where one disk fails, roughly 1/nth of the replicated data will be found on all n of the other disks.
• as the size of the cluster grows failure recovery gets faster.
• FDS uses replication for availability and fault tolerance while recovering data back to stable storage. So it improves durability because it reduces the vulnerability during which additional failures can cause unrecoverable data loss.

Q:- “In our 1,000 disk cluster, FDS recovers 92GB lost from a failed disk in 6.2 seconds.” What is normal throughput of a hard disk? What’s the throughput of this recovery? How can this be possible?
✓ Normal throughput of HD 80-100 MB/S.
✓ Approximately 15 GB/S, very high.
✓ Possible because of large number of replication.
FAILURE RECOVERY STEP BY STEP

Each row of the TLT lists several tractservers & each row also has a version number, assigned by this metadata server.

**Step 1:** Tractservers send heartbeat messages to the metadata server. When the metadata server detects a tractservers timeout, it declares the tractservers dead.

**Step 2:** invalidates the current TLT by incrementing the version number of each row in which the failed tractservers appears.

**Step 3:** picks random tractservers to fill in the empty spaces in the TLT where the dead tractservers appeared.

**Step 4:** sends updated TLT assignments to every server affected by the changes.

**Step 5:** waits for each tractservers to ack the new TLT assignments, and then begins to give out the new TLT to clients when queried for it.

So, When a failure occurs, clients must wait only for the TLT to be updated; operations can continue while re-replication is still in progress.
REPLICATED DATA LAYOUT

- Wkt, k-way replicated system has k tractservers (disks) listed in each TLT entry. The selection of which k disks appear has an important impact on both durability and recovery speed.

- Imagine that we wish to double-replicate (k = 2) all data in a cluster with n disks. So here, cluster will not meet our goal of fast failure recovery because when a disk fails, its backup data is stored on only two other disks (i+1 and i−1). Recovery time will be limited by the bandwidth of just two disks. A second failure within that time would have roughly a 2/n chance of losing data permanently.

- A better TLT has O(n^2) entries. Each possible pair of disks appears in an entry of the TLT. When a disk fails, replicas of 1/nth of its data resides on the other n disks in the cluster.

- When a disk fails, all n disks can exchange data in parallel over FDS’ full bisection bandwidth network. Since all disks recover in parallel, larger clusters recover from disk loss more quickly.
FAILURE DOMAINS

• **Def:**- a set of machines that have a high probability of experiencing a correlated failure.

• Common failure domains include machines within a rack (since they often share a single power source) or machines within a container (as they may share common cooling or power infrastructure).

• FDS leaves it up to the administrator to define a failure domain policy for a cluster. Once that policy is defined, FDS follows that policy while constructing the tract locator table.

• FDS guarantees that none of the disks in a single row of the TLT share the same failure domain. This policy is also followed during failure recovery: when a disk is replaced, the new disk must be in a different failure domain than the other tractservers in that particular row.

Q. “A failure domain is a set of machines that have a high probability of experiencing a correlated failure.” What is the use of the new concept?
CONCLUSION

Flat Datacenter Storage is a datacenter-scale blob storage system that exposes the full bandwidth of its disks to all processors uniformly.

Its recovery from failed disks can be done in seconds rather than hours.

Programmers can pick the most natural model for expressing computation without sacrificing performance.

With FDS, I/O and compute resources can be purchased separately, each independently upgradable depending on which resource is in shortage.

Finally, systems like FDS may pave the way for new kinds of applications.
QUESTIONS?????