L07c. Distributed File System

Introduction:

- A Network File System (NFS) is a LAN-based file system where you have clients distributed over the LAN and file servers on the same network. So instead of a single client accessing its own file system, all the clients will have access to a centralized file system distributed over the network.
- A file server will use caching to quickly serve the clients' needs.
- Some limitations of NFS are:
  - A central file server is a performance bottleneck that affects scalability.
  - An NFS has a limited bandwidth the access the disk, and the file system cache is limited.
- To solve these issues, the Distributed File System was introduced.

Distributed File System – DFS:

- The idea behind DFS is to get rid of the centralized file system and distribute each file across all the server nodes on the network.
- If a client wants to access a file, it contacts all the server nodes.
- Advantages of DFS:
  - No centralization means no performance bottleneck.
  - Distributed management of the data and the associated metadata.
  - The cumulative bandwidth of all the server nodes facilitates faster access.
  - The cumulative cache capacity of the server nodes improves the overall performance.

Redundant Array of Inexpensive Disks – RAID:

- The basic idea behind RAID is to direct the I/O operations of a file to multiple disks. This is called "striping the file to multiple disks".
- The advantage is to collectively get more I/O bandwidth from the group of disks.
- One disadvantage of RAID is the increased probability of failure since the file is split over multiple disks. This is why RAID incorporates using Error Correcting Code (ECC). One disk of the array would be used to store a checksum of the data stored on the other disks so that if a disk failed, the data on it can be recovered.
- RAID advantages:
  - Increased I/O bandwidth.
  - Failure protection by ECC.
- RAID disadvantages:
  - Increased cost.
  - The problem of "small writes": Inefficient to strip very small files.

Log-structured File System:

- After modifying a file, instead of writing the file itself to the system, we write the “change” that has been made to the file as a log record. The multiple changes made to a file is buffered to a data structure called the “log segment”.
- This log segment is then written to the disk. Because the log segment is contagious, a sequential write to the disk would be performed, which is faster than the random writes of the file itself.
- The log segment is flushed to the disk either periodically or when it fills up (because of a lot of file activities).
- This approach avoids the small writes problem of RAID, since these small writes will be combined with other writes over a certain amount of time.
- For a file read, LFS has to reconstruct the file from the log segments, which might add some latency if the file is being read for the first time. In subsequent reads, the file would be present in the cache so there’ll be no latency.
- Overwriting the same data blocks multiple times invalidates some blocks of the log segments on disk, creating holes in the log segments. These holes need to be cleaned up periodically to avoid wasting memory.
- Journaling File System – JFS: A JFS will have both the data files and the log segments. The system applies the changes saved in the log files periodically to the data segments and discards the logs. Hence, reads from JFS do not involve reconstructing the files as in LFS.

Software RAID:

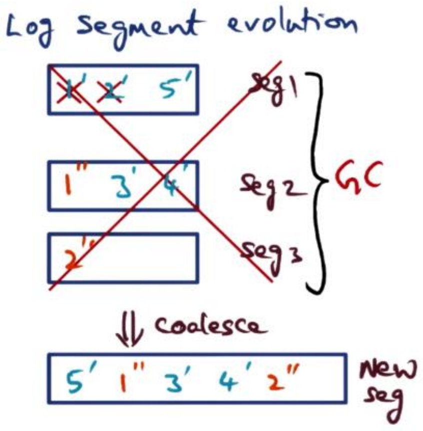
- As mentioned previously, HW RAID has two drawbacks:
  - The problem with small writes.
  - Using multiple hardware disks is expensive.
- Software RAID introduces solutions to these two issues:
  - The Log-structures File System employed by Software RAID solves the small writes issue.
  - Software RAID reduces cost by using the commodity hardware that are associated with the cluster nodes already present on the LAN.
- The Zebra File System, as an example of Software RAID, uses LFS to create log segments, then uses the RAID technology to strip the log segment into the hardware disks of the cluster nodes connected to the LAN.

XFS:

- XFS is another file system that combines the advantages of the previously discussed technologies:
  - Uses log-based striping as in Software RAID (Zebra).
  - Uses Cooperative Caching of flies across cluster nodes: Accessing peer cache memory is faster than accessing local disk (since network communication is very fast).
  - Uses Distributed and Dynamic management of the data and the metadata associated with it.
  - Does not rely on a central server.
  - Uses Distributed Log Cleaning to cleanup log holes.
- Dynamic Management:
  - A centralized NFS will suffer from the following:
    1. The NFS centralized server is not concerned by the semantics of file sharing and is constrained by the its memory capacity for caching files and metadata.
    2. Too many requests for the same file results in hotspots affecting scalability.
    3. Different NFS servers with different loads cannot share their loads.
  - On the other hand, XFS dynamically distribute metadata management over different cluster nodes, resolving the hotspots issue.
  - XFS also employs Cooperative Caching, resolving the limited cache capacity problem of the central server.
- Log-based Striping:
  - Each node will have its own append-only log segment containing the changes made to all the files residing on this node. A change of a file will be stored in a log fragment.
  - Once the log segment fills up beyond a certain threshold (space metric) or if a certain amount of time elapsed (time metric), the log segment and its ECC are then stripped on the different storage servers.
  - The log segment will not be stripped over all the storage servers available so that we don't end up with small write problems. Each log segment will be stripped into a subset of the storage servers (Stripe Group).
  - Stripe Group advantages:
    1. Helps avoiding the small writes problem.
    2. Allows parallel client activities on each group.
    3. Provides increased availability in case of server failure. Because a failure on few disks will only affect some clients.
    4. Different cleaning services can be assigned to different Stripe Groups resulting in efficient log cleaning.
    5. Provide high overall I/O throughput since different subset of servers will be working on different client request simultaneously.
- Cooperative Caching:
  - XFS using the peer cluster memory to reduce the stress of files management.
  - XFS employs a Single-Writer Multiple-Readers protocol to maintain cache coherence.
  - The granularity of cache coherence in XFS is a file block, not the entire file.

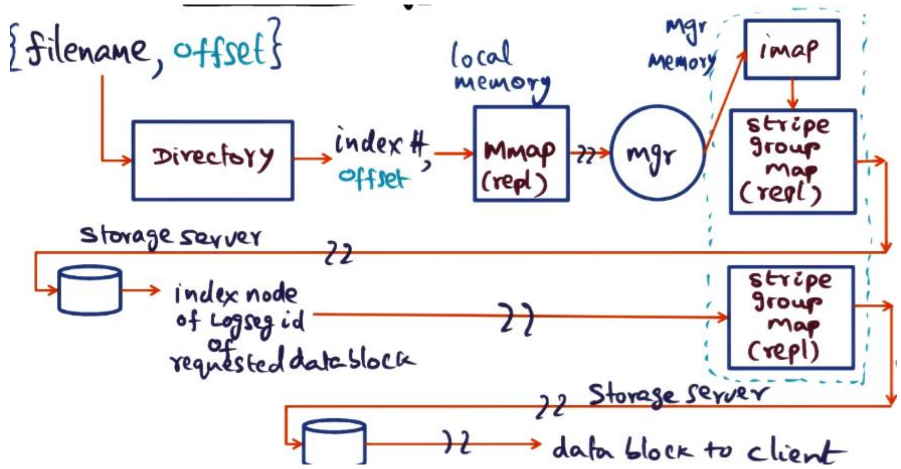
- If a client C_1 requests write access to a read-only file block:
    1. The file manager will send cache invalidation messages to all the cluster nodes that has this file block in their caches to invalidate their local copies.
    2. The manager then provides a write token to the requesting client C_1.
    3. Future read requests for this file block will access the cached memory of C_1.
- In summary, Cooperative Caching satisfies the read requests for a file block from the local copies of this block residing on the caches of the peer cluster nodes, instead of accessing the slow electromagnetic disk.

- Log Cleaning:
    - As the client activities progress in the system, we keep creating holes in the log segments because of frequent modifications to the same file blocks. We need to avoid this wasted space in the system.
    - Log Cleaning is the process of periodically aggregating the live fragments of the log segments into a single new segment to get rid of the dead parts of the log segments (by garbage collection):
        1. Find the utilization status of all the log segments.
        2. Pick a set of log segments to clean.
        3. Read all the live blocks in these segments and write them into a new log segment.
        4. Garbage collect the now not-used log segments.
    - Since we deal with a distributed system, the log cleaning process happens in parallel with writing to the log segments.
    - XFS makes log cleaning a distributed activity by making the clients/stripe groups responsible for log cleaning.
        1. Each client is responsible for collecting the utilization information of the files it's writing to.
        2. Each Stripe Group is responsible for the log cleaning activities of the files residing in this group. Each Stripe Group has a leader that assigns cleaning activities to the members of the group.



XFS Implementation:

- The metadata management on XFS is not static, it's distributed over all the cluster nodes in the system. This is how it works:
  - Each client has a replica of a data structure called the Manager Map (m-map), which is a lookup table that contains the mapping from a file name to its corresponding metadata manager.
  - After consulting the m-map and know which metadata manager node is responsible for this file, the client contacts the node and goes through these steps:
    1. The manager looks up a data structure called the File Directory to get the i-node number, which is the starting point for looking up the contents of the file.
    2. Then using the i-map data structure the i-node number will be mapped to an i-node address, which is the i-node address for the log segment associated with this file name.
    3. Using the Stripe Group Map, we can locate the storage server that contains the index node of the log segment ID that is associated with this file name.
    4. Once again, we use the Stripe Group Map to get the storage servers that contain the data blocks that corresponds to this file name.



  - Caching is employed to make sure that this long path is not taken for every file access. There're three scenarios that we can go through:
    1. The data blocks for the files are already cached locally in the client's memory. This is the fastest path.
    2. The data blocks are not locally cached but are present on a peer cluster node's cache memory. Then the data blocks can be fetched from the cache memory of this node.
    3. If the data blocks are not present on a peer cluster node's cache memory, then we go through the above-mentioned path.

- What happens when a client writes to a file?
  - At the high level, the client writes to a Stripe Group and notifies the Manager about the latest status of the file.
  - At the low level:
    1. The client aggregates all the changes to the files into the log segment data structure in memory.
    2. After crossing a space or a time threshold, this log segment data structure is flushed to a contiguous portion in a Stripe Group of Storage Servers.
    3. Finally, the client notifies the latest status on these modified files to the manager so that it has up-to-date status about the files it manages.