Flat Datacenter Storage

Key Takeaways

- Flat data storage allows any computer to access the data stored by any computer, hence it does not have the constraint of data locality, i.e. compute on the machine that contains the data.
- Flat data storage achieves similar performance with system exploiting data locality. This is because if a slow machine exists, the task can be easily reallocated to other machines without moving a lot of data. The distangling of CPU and storage also makes it easier to balance the usage of CPU and disks so that neither of them becomes a bottleneck.
- When a node fails, flat data storage achieves higher efficiency in recovery since the data can be transferred from many other disks to the new backup disk.
- Flat data storage achieves better performance than previous benchmarks, and state-of-the-art result in sorting.

Flat Data Storage (FDS)

- Flat data storage means data stored by any computer can be accessed by any computer, hence there is no need to consider putting the data in the right place first.
- The flat data storage spares the trouble of thinking data locality, here data locality means the computation should be done on the machine containing the data. In some cases the need to data locality hampers the performance, since it can cause the mismatch between the IO and CPU usage on the machine, and the preference of locality makes it harder to redo the task on another machine since the other machine does not contain the data.
- Flat data storage is built upon the assumption that the network bandwidth at a data center is no longer a scarce resource, which is realized by many commercially available switches.
- Even if the data indeed resides on the same machine for computation, the data are still treated as remote, hence there is no concept of data locality for FDS.
- Even without data locality, FDS can achieve performance similar with system making use of data locality.
- One reason FDS has high performance is it can match the usage of CPU with usage of IO, and a slow job can be retasked easily.

Design Overview

- Data are stored in blobs which are byte sequences named with 128-bit GUID.
- Blobs are divided into tracts which are the basic unit of read and write. Tracts are sized so that random and sequential access achieves similar throughput.
- Every disk is managed by a machine called tractserver that services read and write requests. The tractserver access the disk directly using disk interface instead of file system.
- The write operation is guaranteed to be atomic: it either commits or failed completely.
- By spreading a blob's tracts over many tractservers and issuing many requests in parallel, many tractservers can begin reading tracts off disk and transferring them to the client simultaneously.
- A metadata server contains a list of the system's active tractservers and distribute the list to all clients. This list is called tract locator table(TLT).
- When a tract from a blob is to be read, the tractserver is calculated by hashing the tract ID and the blob GUID. The hash function also randomizes the blob's starting point in the table, to ensure clients better exploit the available parallelism.
- After the tractserver is located from TLT, the read/write requests are sent to the corresponding tractserver from clients.
- 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 tractserver.

- Each blob also has a metadata containing information about the blob such as its length. The metadata is stored in a special tract of the blob as well as on the tractserver. When a blob is updated, its metadata on the tractserver is also updated.
- FDS can dynamically allocate the work to workers, because it does not the contraint of locating computation on the machine containing the data, hence its tasks can be reallocated to any other worker instantly. If the job is sliced fine enough, the finishing time of job only depends on the slowest worker to complete a small percentage of the job since only a small fraction of the job is allocated to this worker. This reduces the problem of straggling worker meaning a slow worker drags down the efficiency of the whole system.

Failure Recovery

- If a disk fails, all other disks containing the replicates sends the data to a new disk, hence the recovery becomes faster when the data system grows larger.
- Since the data are replicated, when a write happens all disks containing the replication will be notified. On the other hand, a random disk will be accessed when read happens.
- For writes that also modifies the metadata, one single primary tractserver is notified. This tractserver modifies the metadata then executes a two-phase commits with the other replicas.
- Each row in TLT contains multiple tractservers since there are multiple replications of the row.
- Each tractserver sends heartbeat to the metadata server. If the heartbeat messages stops, the metadata server considers the tractserver dead.
- When a tractserver is dead, the metadata updates the corresponding affected entries in TLT as well as the version number of such entries. The version number is updated as a flag of change of metadata. If any client request uses the stale version number, the request will be rejected and the client will ask for updated version number.
- Some machines are more likely to fail together, such as machines on the same rack as they share the same power supply. This is called a failure domain. FDS ensures none of the disks for a given entry are within the same failure domain.
- When a new tractserver joins the system, some entries from TLT that were assigned to other tractservers will now be assigned to this server, and the data will be copied to this new tractserver.

Network

- FDS gives each storage node network bandwidth equal to its disk bandwidth, preventing bottlenecks between the disk and network.
- FDS uses a full bisection bandwidth network, preventing bottlenecks in the network core.
- FDS gives compute nodes as much network bandwidth as they need I/O bandwidth, preventing bottlenecks at the client.

Performance

- When the number of client is small, the tractserver bandwidth is much larger than the client bandwidth, hence the throughput is linear to the number of clients. Then the throughput becomes flat when the number of clients is very large and the tractserver bandwidth becomes a bottleneck.
- FDS achieves throughput that is approximately half of local throughput for both read and write.
- When a tractserver with data of about 1TB fails, the failure can be recovered on the scale of 1 minute.
- When FDS is used for sorting data, it achieves state-of-the-art throughput even though using less resource than previous record. A large contribution comes from eliminate stragglers due to dynamical work allocation.
- When used for cointegration data analysis, the FDS achieves 6x speedup compared with public cloud. By compressing the data, the task can be converted from IO bound task to CPU bound task.

- For serving the index of internet for search engine, the FDS increases the speed by 2 using 1/3 of the machine.