L09a. Giant Scale Services

Generic Service Model of GSS:

- Many (embarrassingly) parallel requests are issued to the Giant Scale Service.
- The backend servers are connected by a high bandwidth communication backplane.
- These requests are fielded by the Load Manager which handles the following:
  - Maintains load balance of client traffic for effective server utilization.
  - Provides high availability by hiding partial failures.

Computational Clusters:

- Advantages of Computational Clusters:
  - Scalability: The ability to incrementally add more resources without worrying about rearchitecting the internal structure of the data center.
  - Cost: The ability to easily mix and match hardware of different generations.
  - Performance: The ability to get improved performance by adding more nodes.
- 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.

Load Management at Network Level:

- Round-Robin Domain Name Server:
  - Each DNS request to a domain name is redirected in a round-robin manner to a different IP address to balance the load. The idea behind this approach is that the data is replicated across identical servers.
  - The advantage of this model is that the Round-Robin DNS can redirect the incoming request to the least loaded server.
  - The disadvantage is that it cannot cover down server nodes from the external network.
- Transport Layer Switches:
  - Layer-4 transport-level switches can be architected as switch-pairs so that if a node failed, we can jump from a failed switch to another working switch at the load balancer level.
  - Provides the opportunity to dynamically isolate failed server nodes from the external network.
  - Service-specific front-end node functionality: Facilitates assigning specific nodes to serve specific requests.

- Device-specific characteristics: Facilitates redirecting requests coming from specific devices (e.g. mobile phones) to specific nodes.
- Data is Partitioned to improve Performance: When data is partitioned across servers, each partition can be processed for queries in parallel, thereby improving performance of a particular query.
- If a particular server is down, then there is data loss for the query and hence data coverage is lost. This is why we might have several replicas of the same partition on different servers. Data is replicated for higher availability.

DQ Principle:

- Terminology
  - Q_0: The load offered to the server. In other words, the number of requests incoming to the server per unit time.
  - Q_c: Requests completed by the server per unit time.
  - Q: The server yield $Q = \frac{Q_c}{Q_0}$.
  - D_f: The full data set required to handle an incoming query to the server.
  - D_v: The available partitions of data that can be used to serve the query.
  - D: The Harvest $D = \frac{D_v}{D_f}$. A harvest of 1 means the full data partitions is available to serving the incoming query (no down nodes).
- Definition: For a given server capacity. The product of Harvest (D) and Yield (Q) is constant $(D \times Q)$
  - This means that there's a tradeoff between the number of clients that we're serving, and the amount of data used to serve the requests.
  - DQ works under the assumption that in Giant Scale Services, system performance is bound by the network capacity not the I/O capacity.
  - The System Administrator can use the DQ principle to deal with reduces system capacity by either sacrificing Yield of Harvest.
- Uptime vs DQ:
  - Another metric used to measure the server performance is the Uptime.
  - We calculate two values:
    1. Mean Time Between Failures (MTBF): The mean time between 2 failures, measured over many failures.
    2. Mean Time To Repair (MTTR): The mean time to repair a failure, measured over many failures.
  - The Uptime is then calculated as the ratio of time that a server was up between failures:
  $$Uptime = \frac{MTBF - MTTR}{MTBF}$$
  This will be a number between 0 and 1, and it's typically larger than 0.9.
  - The uptime is not a good measure of how the server is performing since it takes into account times where there were no requests to the server at all. Hence, DQ is a better measure.

- The DQ principle will provide the System Administrator with the information to decide:
  - How much data to replicate?
  - How much data to partition?
  - How to deal with failures?
  - How to gracefully degrade the server when the volume of incoming traffic increases beyond the server capacity?

Replication vs Partition:

- When data is replicated, if a failure happens, Harvest (D) will remain unchanged, while Yield (Q) will decrease.
- When data is partitioned, if a failure happens, Harvest (D) will decrease, while Yield (Q) will remain unchanged.
- When the full corpus of data is not available for the case when data is partitioned and there are failures, then the Fidelity (Quality) of results will be less than that when the full corpus of data is available.
- The DQ is independent of whether we are replicating or partitioning the data. Remember that the underlying assumption in the DQ principle is that the Giant Scale Services are Network-bound and not I/O-bound.
- In the rare condition when there is a significant write traffic to disk (i.e. requests are I/O-bound and NOT network-bound), replication may require more DQ than partitioning.
- Beyond a certain point, a good strategy is to both replicate and partition the data.

Graceful Degradation:

- When there are failures or server saturation, the DQ principle is very useful in managing graceful degradation of the service from a client's point of view (instead of sudden failures that are visible to all users).
- We have two possibilities:
  - Keep the Harvest (D) the same. Every client request has the same Fidelity in terms of query results. In this scenario, the Yield (Q) decreases.
  - Keep the Yield (Q) the same. Keep the volume of served clients the same. In this scenario, the Harvest (D) decreases.
- DQ provides an explicit strategy to manage saturation as a tradeoff between fidelity and availability.
- Another option available for the System Administrator is to use a cost-based admission control. Pay more to get more.
- Another option is to use a priority-based admission control. Critical/important applications will be given higher priority.

Online Evolution and Growth:

- As the services are evolving continuously, the online evolution and growth of GSS is handled by managing the DQ loss.
- There’re three choices in managing service loss during server upgrade:
  1. Fast Reboot:
     - Bring down all servers at once, upgrade all of them and then turn them back on.
     - The Fast Reboot approach is particularly useful in situations where the user community is segmented such that the fast reboot can be performed during off-peak hours (say night time).
     - The whole user community will experience service unavailability for a very short period of time.
  2. Rolling (wave) Upgrade:
     - Rather than bringing all the servers down at the same time, bring down one server at a time, upgrade it and then do the same for the next server and so on.
     - Rolling Upgrade is done for batches of servers and takes longer time than Fast Reboot.
     - However, Rolling Upgrade has the benefit that service is partially available during upgrade as compared to Fast Reboot where the service is completely unavailable during upgrade.
     - Different segments of the user community will notice service unavailability for a short period of time.
  3. Big Flip:
     - Bring down half the number of nodes at once.
     - The service is available at 50% of the full capacity.
     - The Big Flip upgrade time is more than Fast Reboot but less than Rolling Upgrade.
     - Big Flip reduces the total DQ capacity available by 50% during the upgrade time.
     - Half the user community will experience service unavailability.
- Total DQ loss for online evolution:

$$DQ \times n \times u$$
$$n = number\ of\ nodes$$
$$u = time\ to\ upgrade\ one\ node$$