**Description of Data Storage**

A storage engine is an internal software component that a data store uses to store, read, update and delete data in the underlying memory and storage system.

Key features; the following features will be part of system requirements:

* Related to the storage system
  + distributed B+Trees and LSM for reading, writing, and updating
  + persistent data; possibly on non-volatile memory
  + content-addressable data
  + data versioning (to strengthen collaboration)
    - no in-place updating and deleting of data
    - all data objects recorded will remain in the data store
    - each data object will be versioned
  + maybe compression
  + will have to choose one or several data models (key-value, document, graph and wide-column)
* Related to distributed systems
  + replication, concurrency, and consensus
  + membership (with gossip style)

**Distributed Systems Implementation**

1. Replication vs Sharding
   1. Terminology

Partitioning or Sharding?

* <https://qr.ae/pGaPSz>
* <https://qr.ae/pGaPJa>
  1. Sharding

Thus, I was thinking of Sharding data.

A partitioned system can be adjusted to handle a larger workload by creating more partitions and distributing them on more machines.

In this approach, the system can also use relatively low-spec machines, which are typically more cost-efficient.

Scalable, not sure about fault tolerance; because if a node that contains a specific partition is unavailable, then it does not sound fault tolerant to me.

**Challenges:** When we partition the data, the system needs to maintain bookkeeping of which part of data is stored in which node. Based on this information, the system needs to query the corresponding node when a request is made, in order to guarantee high performance.

Efficient indexing schemes to efficiently access data that’s been sharded.

Figure 1: Sharding data

* 1. Replication

To achieve fault tolerance, I’d implement replication.

Replication is the process of creating multiple copies of the same data and storing them into multiple machines. Each one of these copies is typically called a replica.

Primary-backup replication and consensus-based replication.

If we go the P/B Replication way, then no need for Consensus-implementation?

**Challenges:** As explained before, different replicas contain the same data. This means that any changes to the dataset need to be performed on all the replicas in a controlled way. Otherwise, there is a risk of having different data on each one of the replicas.

Figure 2: Replicating data

* 1. Dual/Combined Approach (Sharding and Replication)

The common pattern of doing this is splitting the dataset into separate partitions and then creating multiple replicas for each partition.

Figure 3: Sharding data, then replicating those sharded data objects

Will still face the limitations and challenges of both.

* **Sharding (partitioning):** still researching
* **Replication:** Causal consistency model

1. Consistency and Replication Protocols
   1. Causal Consistency

Under the causal consistency model, all processes have to see causally related operations in the same order.

Figure 4: Write operations with no causal relationship

Figure 4 shows an example of causally related writes. In addition to a written value, we now have to specify a logical clock value that would establish a causal order between operations.

P1 starts with a write operation write(x,∅,1)→t1, which starts from the initial value ∅.

P2 performs another write operation, write(x, t1, 2), and specifies that it is logically ordered after t1, requiring operations to propagate only in the order established by the logical clock.

Figure 5: Causally related write operations

This establishes a causal order between these operations. Even if the latter write propagates faster than the former one, it isn’t made visible until all of its dependencies arrive, and the event order is reconstructed from their logical timestamps. In other words, a happened-before relationship is established logically, without using physical clocks, and all processes agree on this order.

Figure 6: Write operations with causal relationship

Causal consistency can be implemented using logical clocks [LAMPORT78] and sending context metadata with every message, summarizing which operations logically precede the current one. When the update is received from the server, it contains the latest version of the context. Any operation can be processed only if all operations preceding it have already been applied. Messages for which contexts do not match are buffered on the server as it is too early to deliver them.

A vector clock is a structure for establishing a partial order between the events, detecting and resolving divergence between the event chains. With vector clocks, we can simulate common time, global state, and represent asynchronous events as synchronous ones.

1. Consensus
2. References (Sources)
   1. <https://dimosr.github.io/partitioning-and-replication/>
   2. <https://raghumb.gitbooks.io/a-guide-to-software-architecture/content/database_concepts/replication_partitioning.html>
   3. <https://qr.ae/pGaPSz>
   4. <https://qr.ae/pGaPJa>