Database Fundamentals

Mo D Jabeen

September 5, 2022

1 Evolution of database design

1.1 What has triggered the evolution in database?

- Business need to be agile, hypothesis/business models need to be tested fast and then decisions made if a pivot is required. Market insights should allow for quick changes to products/operations.
- CPU improvement is decelerating, and parallelism is increasing

1.2 What is data intensive application?

A data intensive application primary challenge is the use of data (storage, transformation, transmission etc) this is primary bottle neck whereas in compute intensive apps CPU cycles is the bottle neck.

2 Fundamental metrics

Reliability: Tolerate hardware,software and human faults.

Fault and failure are different.

- Fault: Deviates from specification
- Failure: Where the system stops working

Better to build fault tolerant systems that don't lead to failure. Occasionally (ie system security) better to be fault preventive.
Worth creating faults as a testing methodology and questioning any assumptions the code bases uses to run successfully. Systematic faults are often caused by code assumptions which are true most of the time.

2.0.1 How do you prevent human error?

- Allow fast roll backs to default configs
- Allow for data to be recomputed
- Setup good telemetry (monitoring of system health)

Scalability: Maintain load and performance as quantities increase.

2.0.2 How do you determine the focus on when increasing scalability?

Initial focus should be on determine the most intrinsic load parameter to the current architecture (ie database size, size of average reads etc).
Then question what is the effect on performance given the current resources as the load parameter is increased OR how would the resources have to be changed.

2.0.3 How do you measure performance?

The most releavant performance metric should be determined (ie throughput, response time latency.)

This metric will most likely have a distribution even given local consistency due to external factors. The mean of this value should generally be ignored as it does not represent an actual value experienced by the system. Instead the median is an effective tool and the use of percentiles (50th,95th,99th etc).

2.0.4 How to scale?

Common knowledge is to scale on a single node before moving to a multi node setup as it is simpler. This is until it is worth the cost of the change. Better to focus on the ability to iterate quickly than scaling for a unknown future load.
Maintainability: Operability, simplicity and evolvability. (Ease of understanding).

The majority costs of software is upkeep not development. There are three main elements:

- **Operability:** Easy to operate
- **Simplicity:** Easy to understand
- **Evolvability:** Easy to change

2.0.5 How do you enable good operation?

- Good monitoring
- Standard tools
- Good documentation
- Predictable behavior
- Avoid single machine dependency
- Good default behavior

Accidental complexity: Arises from complexity of the implementation and is not inherent in the problem being solved.

3 Database Systems

3.1 What are some elements of a database system?

- Cache - Results of expensive operations
- Stream processing - Asynchronous processes messaging
- Batch processing - Crunch a large amount of accumulated data
- Message queue - Hold data for use with other processes

3.2 Which tool should you use?

No single tools fits all applications, instead the work should be broken into tasks and then the most appropriate/effective tool used.
Example: Caching - Memcached

4 Internal vs External

4.1 In code data structures

Data structures in code are should be structured and used differently to external databases. In code data ie for OOP should be based around their use in logic. Databases can be used by multiple separate processes whereas data structs should only be used by local code.

Data structs should:

- Be limited in size
- Not generally used for concurrent programs
- Not tied to ACID (Atomic,consistent,isolated and durable)
- Fast

5 Understanding Text

5.0.1 subsubsection