Database Fundamentals

Mo D Jabeen

October 12, 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 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 Data Representation

Data representation is crucial as it will directly influence how a problem is solved. The choice of method of storage is key:

- JSON
- XML
- relational db
- graph models

The most popular current db model is SQL, which is based on using relational databases via relational database management systems.
NoSQL was born from:

- Need for greater scalability with a large writing throughput
- Desire for more dynamic and expressive data modelling (schemas can be static)

5.1 How to handle many to one relationships?

1. Ref to another table with a foreign key
2. XML or json to have multiple entries per row
3. Refer to a XML or JSON doc in the entry

In relational dbs can use **joins** to combine related data, however more difficult to handle with document or NoSQL dbs. As an apps features grows more data is collected with key links to old data increasing the **many to many relationship issue.**

The key issue is to solve the many to many relationship without duplication (lacking normalisation).

5.2 What are document databases good for ?

The data is normally stored in one continous string, so if all the data is loaded into the app at onces, have great locality. However normally recommended to separate docs as if any changes are needed to that cause the file size to change the entire file will be rewritten.

Have a very flexible schema, allowing the use of heterogenous data ie

- Each object requires a different table
- Data is from external sources and its format may change

Not so good at dealing with many to one or many to many relationships.

5.3 Declarative or imperative lang?

SQL is declarative, the compiler will decide how to implement the requirements. An imperative lang dedicates how to perform the requirements of the code (ie the sequence the code is executed).

5.3.1 Why is declarative useful ?

Good at abstracting away the db implementation, allowing the RDBM to optimize without needing to changing queries. Also great for parallel computing.

5.3.2 How can you use SQL features with NoSQL?

NoSQL is implementing methods to allow working easily with subsets of data ie map and reduce. The two different methods are converging.

6 Storage and retrieval

6.1 Appended files

6.1.1 What is the simplest method of storage ?

Simplest most effective method is to use an append log for storage.

6.1.2 How do you speed up retrival ?

Without parsing through the document, retrieval is speed up by using an index.

However there is a trade off, as index updating will slow down your write speed.

6.1.3 How does a hash index work ?

Store all keys to the hash in memory allowing for quick looks ups. Will cause issues if the memory size issues with a large key set.
Works best with a small set of keys that are often changed.

Con: Difficult to work with range queries, as index are not stored sequentially.

6.1.4 *How do you manage the size of the append log ?*

Segment the files compress and remove any duplication. The segmented files can be concurrently merged back.

6.1.5 *What is the standard search approach for finding a key?*

Search the most recent segment, then the next most recent etc.

6.1.6 *How are delted entries managed?*

A special message is appended into the log, which is used to ignore the chosen entry at the next merge.

6.2 LSM / SS Tables

6.2.1 *What if you sort the key value pairs ?*

In hash indexes the key value pairs are stored sequentially. In a Sorted String table (SS Table) the key value pairs are sorted by key. Can then make assumptions about the location of a value, due to the ref values and the expected order. Can take these segments between ref values and compress them for better bandwidth.

6.2.2 *How is a LSM tree created ?*

1. Write to in memory structured tree, by key.
2. When big enough store data in a SSTable sorted by key
3. Read data by first checking the mem table then the SSTables in disk
4. Merge and compress segments on disk in the background

6.2.3 *What are issues with LSM trees?*

When writing the memtable to disk a cash would cause lost data, so append logs are used to store data in the memtable deleted when memtable stored as SStable.

6.2.4 *How do you optimize finding a key?*

Use **Bloom filters** to quick check if a key is in a segment.

6.3 B trees

6.3.1 *What is the most standard index implementation?*

B trees are the most widely used and standard index implementation. They use fixed size blocks called pages. LSM are appended B trees changed in place. Fixed size implementation matches how storage is done in hardware.

6.3.2 *How are B trees built?*

Pages are linked to each other, some pages refer to child pages via a ref key value and the boundaries between the given references show which which child page will hold the key being searched for.
Some pages will have the actual key and its value for indexing, these are called leaf pages. The number of refs in a page is called branching factor.

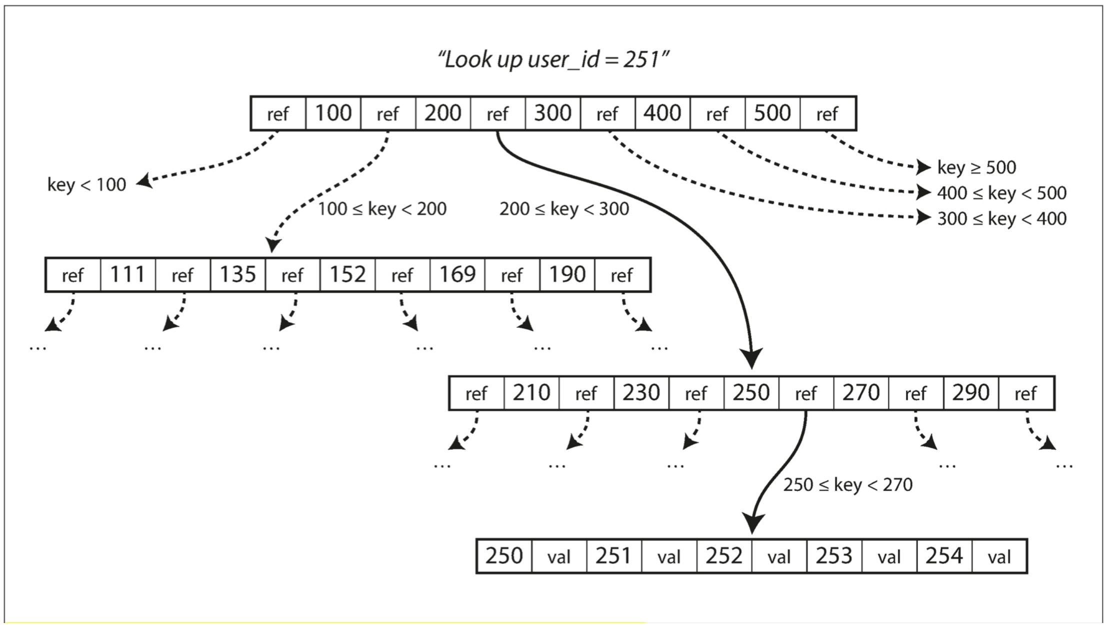
"Look up user_id = 251"
ref 100 ref 200 ref 300 ref 400 ref 500 ref
key < 100
100 ≤ key < 200
200 ≤ key < 300
key ≥ 500
400 ≤ key < 500
300 ≤ key < 400
ref 111 ref 135 ref 152 ref 169 ref 190 ref
ref 210 ref 230 ref 250 ref 270 ref 290 ref
250 ≤ key < 270
250 val 251 val 252 val 253 val 254 val

Figure 6.1: B trees page construction

If a page does not have enough space for a new key, its split into two and the references are updated.
A four level tree with 4kB pages and a branching factor of 500 can hold 256 TB of data. The actual data stored is structured into the tree.

6.3.3 *What are the issues with B trees ?*

When splitting a tree and updating the references is there is a crash in between can create corrupt data. So write ahead logs are used, a file that appends changes before they are executed. They also need to be careful when dealing with concurrency due to the required in place writes. So use latches (light weight locks).

6.3.4 *How can B tree performance be improved?*

Write a new modified page and change the reference to it, this will be more performance effective than modifying a page and keeping a write ahead log.

6.3.5 *What is better LSM or B trees ?*

Rule of thumb:

- LSM has better write throughput
- B trees have better reads

However many optimizations to B trees to improve its write throughput.

7 Transactional vs Analytical Data

7.0.1 *What are Transaction and Analytical processes?*

Transaction process track every day writes with updates happening often. Allows for low latency small reads and writes. **OLTP - Online Transaction Processing**.

OLTP data is crucial for business so queries and bandwidth is tightly controlled.

Analytical processes are large data amounts queried and uses for analysis. Generally these are instead stored in a date warehouse, which uses very different patterns for indexing (Not B or LSM trees). **OLAP - Online Analytical Processing**.

7.0.2 How are data warehouses created ?

The data warehouse includes copies of the data in the OLTP data that has been extracted, transferred and then dumped into the database. **ETL - Extract Load and Transform**
They most commonly use relational databases ie sql.

7.0.3 What is star and snowflake schema ?

Star schema: One main fact table then many dimension tables ie product SKU, date ID etc
Snowflake schema: Has many layers of dimensions.

7.1 What is column storage ?

Instead of storing all values from each row together, instead store all values from each col. Therefore queries only need to query and parse the values used in the query. This is bad for OLTP as you will need to look and edit many files/tables for every transaction.

7.1.1 How do you compress column storage ?

Bitmap encoding is a form of compression, where one bitmap is created for every distinct value and one for each row. If that row contains that value the bit is 1 or 0 if it does not.
Good for smaller number of distinct values used. Can be good for CPU cycles too as compressed data can be perform bit operations directly.

Sorted cols can be used for faster queries ie sort by most queried.**ALl columns must use the same order for all files so a row can be reconstructed.**

Can also create summary "materlized view" tables allowing for quick queries when comparing two dimensions, such as sum.

Colum storage uses a similar concept to LSM trees for storage, hold in memory then written to disk. B trees are very hard as doing in places changes over many files will be very difficult.

8 Encoding and Evolvability

Changes to the schema caused by new features and data. Old and new code should be able to deal with these changes. Should also allow staged rollouts for better evolvability with small frequent changes.

8.0.1 What is the difference between in code data and data sent over the network ?

In code the data is stored in the programs structures, which are optimized to work with the CPU and use a pointer to access.

Over a network you need a self contained sequence of bytes ie JSON sent using some type of protocol ie HTTP.

8.0.2 How do you convert from memory to network data ?

Encoding, marshalling and serlization are terms for changing program data to byte sequences.The reverse is decoding, unmarshalling and deserilization.

8.0.3 What is the issue with the libarires used by langs to encode/decode?

- Program specific
- Allows arbitrary classes to be instantiated (which is a security issue)
- Bad performance/compatibility

These should be only used for a small period of time and small issues.

8.0.4 What are the types of Encodings?

The most standard and popular are JSON and XML (which is considered too complex).

Binary encoding are more space efficient, however are not in a human readable format.When TB of data is being sent this will matter. Apache thrift, protocol buffer and avaro (particularly flexible schema) are examples of binary encodings. They also gave a much simpler and less strict schemas rules giving better evolvability.

8.1 Modes of data flow

8.1.1 What are potential issues with using a database as a dataflow ?

Will often have many processes accessing the db at any given time, its important that different version of code work with the database. New data for a col should work with old and new code.

Forward compatibility is particularly important, there is a need to be careful of lost data in old code collecting data from the db converting it to an object then converting back to be placed into the db!!!

Migrating to new schemas is expensive so instead old data is left and new data is fitted into the db, ie use a default null val to a new col.

8.1.2 How is data sent over a network?

Severs that expose data request via an API are called services. APIs use an agreed standard set of protocols ie HTTP,URL, TLS etc. Web apps uses services and a JavaScript app using XML-HTTPRequest to become a HTTP client is known Ajax.

Services are similar to a database allowing input and output of data, however they are very restricted by the business logic giving encapsulation.

8.1.3 What are the HTTP API methods ?

The two most popular are REST and SOAP. REST is a philosophy based around the tools of HTTP. SOAP is XML based and is trying to be HTTP independent, uses a bunch of features from the web service framework. Also uses an XML lang which can be good for statically typed languages.

RESTful APIs are more popular and use Swagger as a definition format.

8.2 What about RPCs ?

Remote procedure calls (RPCs) are formed on the idea of making network calls feel like calling a local function. Which is called location based transparency.

8.2.1 What is bad about RPCs?

Local function and remote functions are not the same:

- Network may return no result ie timeout
- Retrying may make a process run many times in a service
- Network calls will differ in latency
- Program data sent as encoded bytes can be problematic

8.2.2 Examples of RPCs ?

- gRPC uses protocol buffer
- Fingle uses Thrift
- Rest.li uses JSON over HTTP

Binary encoded RPC have better performance than generic methods ie JSON using REST.

8.2.3 What are message brokers ?

The middle men used a data queue to make messaging safer and simpler.
Send message to queue or topic, message is sent from topic to subscriber or consumer.

8.3 What are actor models used for ?

Used for concurrency, an actor is an entity that sends messages to each other. As only one message is dealt with the worry of threads is gone.