Why use graphql?

GraphQL offers many benefits over REST APIs. One of the main benefits is clients have the ability to dictate exactly what they need from the [server](https://stablekernel.com/mobile-app-development-services-atlanta/backend-development/), and receive that data in a predictable way.

2. With 1 single API call can hit 2 endpoints, you can have in your api gateway set up as a graphql endpoint that relies on other microservices

Graphql helsp with the overfetching and underfecthing problem

3.

**Graphql nested query depth here**

e. So, it also shows the same problems when a client requests too many nested fields data at a single time. So there must be a mechanism like maximum query depths, query complexity weighting, avoiding recursion, or persistent queries to stop inefficient requests from the client-side.

Caching is a bit harder to set up here.

What’s the difference between high level diagram and low level diagram?

High level

Very genearl system design questions here

1. System architecture
2. Database design
3. Brief description of systems, services, platforms, and relationships among modules.

When do we use sql vs nosql? Then If a lot of relations use sql, follows ACID properties

**Nosql**

1. If data changes structure all the time, use no sql then

2. SQL databases are better for multi-row transactions, while NoSQL is better for unstructured data like documents or JSON. SQL databases are also commonly used for legacy systems built around a relational structure.

3. This scaled very quickly in that sense here

**System design**

**1. Which database should we use for reddit? Sql or no sql**

**What’s load balancing?**

Load balancing is the process of distributing network traffic across multiple servers or network devices to ensure that no single device is overwhelmed with too much traffic. This is done in order to improve the overall performance and availability of the system.

There are several ways to accomplish load balancing, including:

* DNS Load Balancing: This method uses the Domain Name System (DNS) to distribute traffic across multiple servers. When a client requests a specific domain name, the DNS server responds with the IP address of a server that is currently available to handle the request.
* Hardware Load Balancing: This method uses specialized hardware devices, such as load balancers, to distribute traffic. These devices sit in front of a group of servers and use various algorithms, such as round robin or least connections, to distribute incoming traffic.
* Software Load Balancing: This method uses software running on servers or on the network to distribute traffic. This could include using a reverse proxy or a load balancer built into the web server software itself.
* Cloud-based Load Balancing: This method utilizes cloud providers’ services such as Amazon Elastic Load Balancer, Azure Load Balancer, and Google Cloud Load Balancer to distribute traffic across multiple servers.

**To set this up**

It’s actually quite simple, basically have a flash application thatrecives messages sitting on port 5000 and then have

To implement load balancing, you can run multiple instances of this application on different servers, and then use a load balancer to distribute incoming requests evenly across all instances. There are many load balancing solutions available, both hardware-based and software-based, such as HAProxy, NGINX, and Amazon ELB.

What are some loading load balancing algorithms here

* 1. R**Round robin** — handles load in a rotating sequential order.
* **IP hash** — In the hash key is calculated based on the source and destination IP address in a request which decides the specific server to which request should be sent to.
* **Least connections** — Server having the least number of active connections are first sent the requests.

Part 2:

How is round robin diff from message queue?

Message queues are nothing bit temporary buffers placed between users/applications and servers to store the message requests and process them in FIFO order asynchronously until the requests/messages are delivered to the desired server.

Since we have a a requests stuff here. This is where rabbitMq is helpful here.

**Once**  we have msg q then we have the data loader stuff here

Used to deal with stuff over here

Can be used as load balancer here

RabbitMQ is an open source message broker. It uses a publish-subscribe model to route data from publishers to consumers. It is scalable and can be load balanced, acting as a reliable and highly available intermediary.

**What is data consitency**

**Data consistency in database this is very important**

**A single server at the start here**

The example here is w

1. We are building a server facebook for students all across the world

With s single server

If you lose data, that’s it one strike and then it’s gone here.

If a user creates a profile, single copy of data will be stored on a server here.

**How do we reduce the latency here with this exapmle**

1. Add more database servers, 2 servers (in different cities, 1 server for oxford student and 1 server for the Havard)
2. No comm between seach server and there is no data sharing so that’s not good here

**How do we solve communication between different servers?**

1. So when you have multi copies of data

A hand writing on a whiteboard

Description automatically generated

So you have 2 copies of data in each data server, and then when u update the one in US and hten the one in Europe here

**Updating informations in both cities?**

1. When update student C in US, how to update C in Europe as well right away? (In bank system) this can take a while
2. You can use TCP here and
3. What happens if network msg fails?

maybe the Europe server is down, what do you do then? Well, you will never get to know unless you get an acknowledgement, right? So the European server should send back an acknowledgement. That's again, quite simple. If you get an acknowledgement, you know, that the update has gone through and you're sorted,

**What if no ack,**

You can retry until the update happens here, infinite.

Usually what happens next

1. If you have server A, server B.

A drawing of a diagram

Description automatically generated

This is called the 2 general problem here, how does server B know server A req has gone through?

To solve this problem

1. Make A the leader (the only one with write), so only A can do the update to write
2. B gets replicas of the data, B is slave node, now B will read the data here

**Using this we can use the 2 pac consistency here**

Using the 2 phase commit protocol

1. 1 leaderA, multi followers here, and then B, C and D.

A person writing on a whiteboard

Description automatically generated

Send prepare first and then commit request

you send a prepare request to your followers, your followers give you an acknowledgement that, Hey, yes, I got the prepare request. And when you get these acknowledgements, you ask your followers to commit.

When lsave gets prepare (which starts begin)

If the leader fails, then the follower will do a rollback, no commit

**What’s the 2PC consistency model?**

Two-Phase Commit is a protocol used in distributed database systems to achieve atomicity across multiple nodes involved in a transaction. It’s termed “two-phase” because it carries out the commit process in two distinct stages: the “prepare phase” and the “commit phase”.

The protocol involves a coordinator, often the node where the transaction was initiated, and several participant nodes. The coordinator ensures that all participating nodes agree to either commit or abort the transaction, maintaining consistency and integrity across the distributed system.

First phase:

**1. Prepare Phase**

The prepare phase begins when a coordinator initiates a transaction. After making the necessary local changes, the coordinator sends a prepare message to all participant nodes, instructing them to prepare to commit the transaction.

Each participant executes the transaction locally, writes the changes to a log, and responds to the coordinator. If the transaction executes successfully, the participant votes to commit and enters a prepared state. If the transaction fails at any participant node, it votes to abort.

System design how to ensure atomicity between transactions in datasbase?

What’s atomic transaction?

1. *In synchronous replication, the leader node waits for all of the follower nodes to acknowledge receipt of the data change before reporting success to the client.*
2. Can be slow as said for the user so not so good in that regard

Aysnc transaciton

*In asynchronous replic*

*ation, the leader node does not wait for the follower nodes to acknowledge receipt of the data change before reporting success to the client.*

*This means that the follower nodes may not have the most up-to-date copy of the data, and*

*How to compensate this?*

In practice, many systems use a combination of synchronous and asynchronous replication. For example, a database might use synchronous replication to replicate data to a small number of critical replicas, and asynchronous replication to replicate data to other replicas. This configuration is sometimes also called Semi-Synchronous.

How do we handle node failures as said?

**Follower Failure**

Followers are relatively easy to handle in the event of a failure. When a follower fails, it can simply reconnect to the leader and resume replicating data. This process is known as *catch-up recovery*.

**Leader Failure**

Leader failure is more challenging to handle, as the leader is responsible for processing all write requests. When the leader fails, one of the followers must be promoted to the role of leader. This process is known as *failover*.

Implementation of replication logs:

**What’s replica log?**

Replication logs are a key part of leader-based replication, a method of keeping multiple copies of a database in sync. The leader node is responsible for writing all changes to the database, and the follower nodes replicate those changes from the leader.

How does master propagate cahnge to the follower

**What’s the logical way of logging?**

1. Using logical (role-based log replica?
2. Logical (row-based) log replication is a more flexible type of replication than WAL shipping. In logical (row-based) log replication,
3. the leader logs every change to the database at the granularity of a row. The log record contains enough information to identify the row that was changed and the new values of all columns in the row.
4. The leader then sends the log record to the followers, and the followers apply the change to their databases.

And then here we have more code here

1. And then

**Q: How can we achieve strong consistency between the master and slave databases?**

A: WAL (Write-Ahead-Logging), a well-known and important technology in database, is needed for this purpose-. When the system updates the operation write logs (Oracle Redo Log and MySQL Binlog among others) or commits a transaction, one should first ensure the flushing of the logs generated by the transaction to the disk. This would ensure no data loss occurs.

The method for achieving the strong data consistency between the master and slave databases is also simple:

* When you commit a transaction, it initiates two log writing operations: the first one to write the log to the ephemeral disk, and the other to synchronize the log to the slave database and ensure it is saved to the disk
* The master database will return the response to the application only upon the successful return of the two operations, at which point the transaction commitment is successful.

**High availiabitly and what does this handle?**

Why is this important as said?

me time. When it detects any connection failures to the master database, it switches the database service provider to the slave database. The

What about network failure from beofre?

This will keep things ocnsistent here very important

A diagram of a network jitter between a person and a network jitter

Description automatically generated

A: The answer is yes. We can introduce protocols similar to Paxos and Raft to solve the various challenges to ensure the continuous availability of the entire database system. These protocols still use a strong consistency system composed of two databases as the master and slave databases, and HA for monitoring and switching between the master and slave databases.

And then here you have the following:

Q: How to ensure a high performance while ensuring the strong synchronization?

Diagram of a diagram of a computer

Description automatically generated

When app init a req to commit a transaction, must sync the transaciton log to slave db here to ensure robust sync of data and store the log to the disk

System desgin design reddit:

How does CDN work?

1. *A user requests a web page from a website.*
2. *The request is directed to the nearest CDN server, instead of the origin server.*
3. *The CDN server checks its cache for a copy of the requested content. If it finds a copy, it serves the cached content to the user.*
4. *If the CDN server doesn’t have a cached copy, it will request the content from the origin server.*
5. *The origin server responds with the requested content, which is then cached by the CDN server.*
6. *The CDN server serves the content to the user and also stores a copy of the content in its cache.*
7. *Subsequent requests for the same content can be served from the CDN cache, reducing the load on the origin server and improving the user’s experience by serving the content from a geographically closer server.*

Using caching here

1. Talk about the ups and downs of this here and there.

Caching

1. **Cache hit and cache write**

**Basic terms:**

Three concepts that are very important while trying to understand caching are —

*Cache Hit — In simple terms, cache hit is the process in which the user request is fulfilled by the cache i.e the requested file is available in the cache and the moment user requests it, the cache fulfills the request.*

*Cache Miss — When the requested file/data is not present in the cache*

*Cache Write — When cache miss occurs, the application reads the data from the database and writes it to the cache after catering to the user’s request.*

**Caching Strategies and Techniques:**

Overview of Different Caching Strategies:

* In-memory caching: Storing data in memory for ultra-fast access. Commonly used for small to medium-sized datasets.
* Disk caching: Caching data on disk for larger datasets or when persistence is required.
* Distributed caching: Caching data across multiple nodes in a distributed system for improved scalability and fault tolerance.

What are cache eviction policies?

**Cache Eviction Policies:**

* LRU (Least Recently Used): Evicts the least recently used items from the cache.
* LFU (Least Frequently Used): Evicts the least frequently accessed items from the cache.
* Other policies include FIFO (First In, First Out), LIFO (Last In, First Out), and random eviction.

**Cache Invalidation Mechanisms:**

* Time-based invalidation: Setting an expiration time for cached items and removing them after the expiration.
* Event-based invalidation: Invalidating cache entries based on specific events or triggers, such as data updates or changes.
* Manual invalidation: Explicitly removing or updating cache entries based on application logic or user actions.

What are the top mechanims available?

The following is quite important as we see different cache mechanism

**What r diff cache policies and their advantages**

Two common approaches are cache-aside or lazy loading (a reactive approach) and write-through (a proactive approach). A cache-aside cache is updated after the data is requested. A write-through cache is updated immediately when the primary database is updated. With both approaches, the application is essentially managing what data is being cached and for how long.

Cache aside or lazy loading:

1. Check the cahce to see if available

2. If the data is available (*a cache hit*), the cached data is returned, and the response is issued to the caller

3.. If the data isn’t available (*a cache miss*), the database is queried for the data. The cache is then populated with the data that is retrieved from the database, and the data is returned to the caller.

Advanatge vs disadavantage here?

* **Resource Efficiency**: Conserves resources by loading data into memory only when it is needed.
* **Faster Initial Load Times**: Reduces the time required for an application or webpage to become initially responsive.
* **Improved User Experience**: Enhances user experience by prioritizing the loading of essential content first.

Disadvantage

* **Potential Latency**:Introduces latency as data is loaded on-demand, impacting real-time responsiveness.
* **Complex Implementation**: Implementation can be complex, especially for large or intricate systems.
* **Could also take longer as said here**

Disadvantage here

* **Higher Write Latency**: May introduce additional latency for write operations due to synchronous updates to the data store.
* **Potential Bottleneck**: Write-through caching can become a bottleneck if write operations are frequent or resource-intensive.

Lazy loading Applications here

* **Image Galleries in Web Applications**: In a photo-sharing app, lazy loading can be applied to image galleries. Only images that are visible to the user on the screen are loaded initially, reducing the initial page load time.
* **Infinite Scrolling in Social Media Feeds**:Social media platforms implement lazy loading for infinite scrolling. New content is loaded as the user scrolls down, ensuring a smoot

Applications:

1. Write through better for traditional banking

What’s write thru application?

Write-Through caching is a strategy where write operations are synchronously written to both the cache and the underlying data store. This ensures that the cache and the data store remain consistent, offering real-time updates but potentially introducing additional latency due to the synchronous nature of the write operations.

Advantage:

* **Real-Time Updates**: Provides real-time updates to the cache, making it suitable for applications requiring immediate consistency.
* **Reduced Risk of Stale Data**: Minimizes the risk of serving stale data to users by keeping the cache in sync with the data store.

Disadvantage:

Using this here

1. The application, batch, or backend process updates the primary database.
2. Immediately afterward, the data is also updated in the cache.
3. The application, batch, or backend process updates the primary database.
4. Immediately afterward, the data is also updated in the cache.

Using indexing is very good as said

1. Can improve the code quite a bit

What are indexing important information?

**Indexing Data Structures and Algorithms:**

* B-tree: Implements a balanced search tree structure to enable efficient range queries and sorted data retrieval.
* Hash table: Utilizes a hash function and an array-based data structure to enable constant-time lookups.
* Inverted index: Stores a mapping from terms or keywords to the documents or data points where they occur, enabling fast text-based searches.

**Indexing in Databases:**

Indexing Strategies for Relational Databases:

* Primary keys: Unique identifiers for each row in a table, providing fast access to specific rows.
* Secondary indexes: Indexes created on non-primary key columns to speed up queries that involve those columns.
* Composite indexes: Indexes created on multiple columns to improve query performance for multi-column conditions.

Indexing for Specific Query Types:

* Range queries: Indexing techniques like B-trees are effective for optimizing range-based queries, where a specific range of values needs to be retrieved.
* Text search: Full-text indexing techniques are employed to enable efficient keyword-based searches in textual data, considering factors like relevance and ranking.

What’s vertical sharding vs horizontal sharding