Individual report: Amazone

1.) Personal reflection essay

a) I designed the customers and pastOrders collection alongside 1 teammate. We embedded the current orders within customers as we made the assumption that the current orders collection won't increase by a large amount and we know that this kind of document will require regular querying. Past orders were separated and put into its own collection - this is because these orders are seldom requested and the number of orders can get quite large.

I decided that we shouldn't have longitude and latitude (for all the locations in the collections) as 2 stand-alone fields, instead it should be inside a GeoJSON structure in order for us to be able to do the queries which require calculating distance between 2 points. Here is an example of the structure used:
<field>: { type: <GeoJSON type> , coordinates: <coordinates> }
Where <coordinates> is an array which holds longitude as its first value and latitude as its second.

As part of my work on query 2, I decided we should create an index on the location field within the suppliers collection as it was needed to be able to perform find queries to get the nearby locations based off another collections location coordinates. The type of index was a 2dsphere as this index is normally used in MongoDB to query geographic data.

b) I learned a lot about how to design a NoSQL schema that is optimized based on the requirements and for read/write performance. I believe this project enabled me to better understand the aggregation pipeline with its different stages and allowed me to get more familiar with complex queries that I wouldn't have explored in class. Furthermore, I learned task management and separation of tasks based on the order of the projects completion; the schema had to be completed first before data can be inserted, and queries were only done once every single collection had all the data in, which in hindsight didn't need to be done that way. For example, we could've just had the customers, products and suppliers collection and still been able to do some of the queries.

What was challenging was the way the requirements were abstract and we had to make a lot of assumptions which required a lot of back and forth tweaking. Additionally, creating the sample data and inputting it into MongoDB was very time consuming. Especially when some of the collections had embedded documents and array of embedded documents as this made it difficult to convert the csv data into JSON and so some of the inserting had to be done manually. This was quite tedious due to the brackets etc. and the repetitive nature of the work.

2.)

a) Since the operations are expanding to other EU countries and another data center will be set up in Europe, a multi-leader replication strategy would be suitable. If single-leader is used, the users from the European countries will send requests (e.g place an order) to the leader in the UK based data centre. This will increase latency and defeat the original purpose of setting up the Europe data centre. With multiple leaders, each in its own data centre, users in Europe can do many writes/reads to the leader in the EU data centre and not have to worry about the server response time being slow plus have lower latency.

There may be write conflicts whilst users are using the website (for e.g 2 users selected the same product of which there's only 1 in stock). In this case, Amazone can 'hold' the product in the customers basket for a certain amount of time before releasing it so the other customer can place an order.

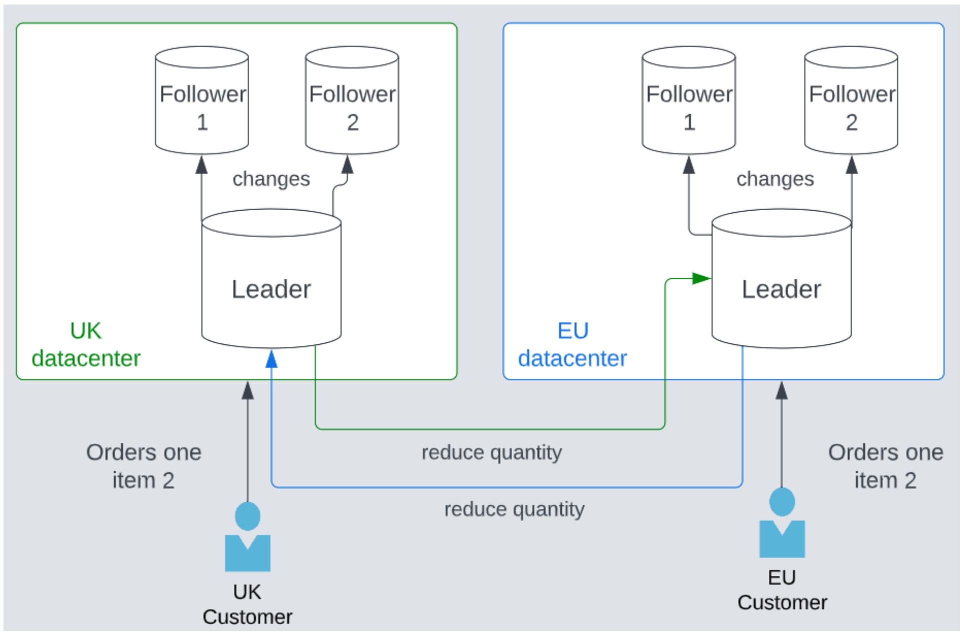

Figure 1 Multi-leader replication in the case of Amazone.

The diagram above shows an example of multi-leader replication in the case of Amazone, where a customer from the UK and EU both place an order for the same product but sending the request to each leader in their respective datacenters. The leader in the UK datacenter has to tell the leader in the EU datacenter that an order has been placed so to reduce the quantity of the product in the database and vice versa. This syncing can cause a conflict; say the quantity was 10 and the EU customer made it 9, now the UK leader won't change the quantity from 10 to 9 as, on its end, the quantity is already 9 (due to the UK customers order). This conflict can be resolved by merging the 2 writes using some algorithm, so that for example the quantities are reduced by 2 automatically instead of applying different strategies (like last write wins) and risking data loss (Verma, 2020).

b) Partitioning allows us to disseminate data so that it is distributed across many nodes. We also want to make sure the data is allocated equally across the nodes. A good partitioning strategy should avoid hot spots – that is, when 1 node takes more (or all) of the load whilst the other nodes stay idle. In our case, partitioning by key-range would be suitable as it would divide Amazones database based on certain ranges. Since we'd want to group the data by geographic location (as we need to keep EU queries separate from UK ones) it would make sense to use a key based on a customers country, or for even more granular partitioning, regions within a country. For example, we could have regions within countries assigned an id (UK-N, UK-S for northern UK and southern UK etc. and FR-N, FR-S for France). If we wanted to get all customers for France, we could query on the partition that has an id beginning with 'FR'.

However, there is a problem that can occur with key-range partitioning and that is hotspots. If there are a lot of customers in 1 particular country, then the load wont be evenly distributed resulting in just 1 of the nodes getting most of the data. We could use key hash partitioning – which applies a function to normally distribute the data across partitions but that would mean giving up the luxury of efficient querying with data ranges. Which I believe is necessary for Amazone as they're setting up a new data center in the EU, so they need to be able to perform queries for geographical regions.

A request routing strategy can be used for allocation of data to partitions. More specifically, a customer could send a request for say, a fresh product, and a routing tier checks the location of the customer and then routes them to the partition that's relevant to the location. E.g a customer in Paris opens Amazone and searches for bananas, but then gets routed to the France website, where data about fresh produce stores are displayed.

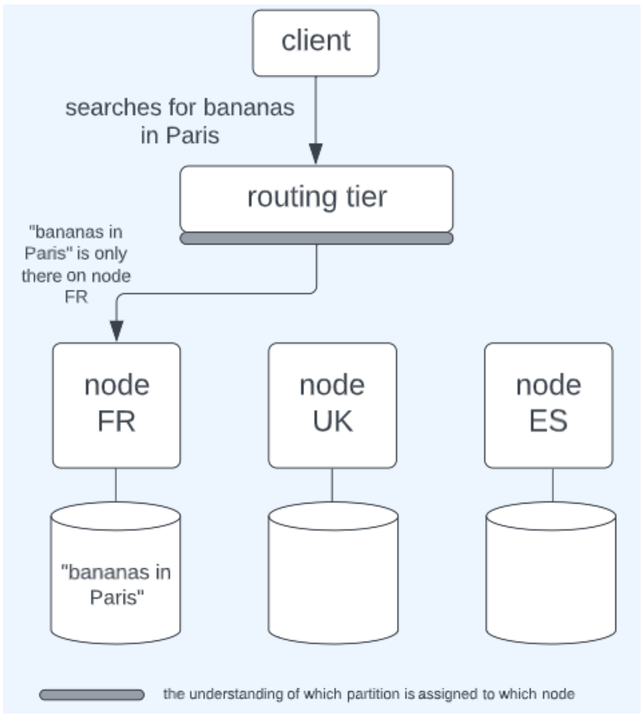
client
searches for bananas in Paris
routing tier
"bananas in Paris" is only there on node FR
node FR
node UK
node ES
"bananas in Paris"
the understanding of which partition is assigned to which node

Figure 2 A very basic example of a allocation strategy that involves a routing tier. Nodes were simplified to FR, UK and ES.

This strategy is very effective as all requests get sent to a routing tier and this knows where every node is so can send partitioned data to its respective nodes. Partitioning in this way also allows Amazone to follow data sovereignty laws and regulations that may vary between EU countries and the UK.

c) The expansion of Amazone to Europe will require them to re-design the database.

- The current database design is not optimized for a large number of users and partners that'd get added on with the expansion to other countries. Indexes can be created on relevant fields (for example country of the user). With this, the database can quickly retrieve data that's needed and improve query performance.
- Different countries have different laws and regulations, meaning the database will need to be designed with this in mind (InCountry, 2021). For example some countries may require users personal information to stay within a particular country. Considering that a multi-leader replication strategy will be used, we must ensure that data that's controlled by law is not replicated and sent to leaders in a different region.
- Behavioral and purchasing patterns are different in different countries, for example one country may not have an affinity to purchasing fresh products online with an instant delivery service and instead still do all their grocery shopping in person. For this country, it would be a waste of resources to have a database optimized for fresh products or the delivery partners who deliver them.
- Amazone will need to support multiple languages in its database for the different countries it enters. They'll also need to store specific information relating to each country.
- The database should be designed in a way which allows for horizontal scaling – with the replication and sharding strategies applied above, it is necessary to have the right keys and fields in order to get the best performance out of these strategies as we can.

In summary, redesigning a database may enhance performance and fault tolerance as well as improving data security while ensuring that it can handle the particular requirements of each region and adhere to local laws and regulations.

References

Sandeep Verma (2020) 'Data Replication in distributed systems (Part-2)' *Medium,* Available at: https://medium.com/@sandeep4.verma/data-replication-in-distributed-systems-part-2-32eba557d78e Accessed (11/01/2023)

InCountry Staff (2021) 'Data Residency Laws by Country: an Overview' *InCountry,* Available at: https://incountry.com/blog/data-residency-laws-by-country-overview/ Accessed (12/01/2023)