Distributed Systems (H0N08A)

Report: Google Cloud 2

Deng Mofan *(r0868382)*
Liang Yinqi *(r0865008)*

December 17, 2021

URL https://true-bit-333719.ey.r.appspot.com

Question 11 The NoSQL database gives up A/C/D of traditional SQL ACID database, also gets rid of the enforcement of consistency. The data itself is not connected, which in turn enables NoSQL to be applied to more different situations easily. The amount of data can be easily extended to huge number and structure of data can be various. NoSQL database provides varies data model including key-values, wide columns and graphics, rather than only a tabular format, and this feature gives possibility for developers to store data as aggregations or simple data structures instead of constructing table data. The data model it provided enables scaling to a large amount of data and a higher user loads, the NoSQL can be operate at any scale. The data scaled horizontally through sharing across servers. The NoSQL database also have high availability because it does not require strong consistency. Instead, it requires eventual consistency. In this case, NoSQL database improves data redundancy so as to significantly improve the availability. While SQL database has to take care of synchronization and transaction operations regarding to redundancy design due to strong consistency, which may cost even more and reduce availability.

Question 12 We create four collections listed as below. shows: showID as key, company, image, location, name, and a list of seats ID as value seats: seatID as key, available, name, price, time, type, seatID,as value tickets (this is for the local company):ticketID as key, company, customer, seatID, showID, ticketID as value bookings: bookingID as key, customer, bookingID, a list of tickets, time as value Most of the data models are the same as backend. However, we think that the system will frequently search seats. Thus, if we put the seats inside the shows as backend data model presents, it will be really slow due to a serious of queries operation. Instead, we only store seatID in shows and the return uses seatID as key to directly get shows, without needing to do complex query.

Question 13 The strong relationships of relational data make it possible to make complex queries with SQL select statements. But in NoSQL we only need to do some simple queries, instead of the deep and complex one. Thus we sometimes need to divide the queries into several steps.

Question 14 In level 1, we rely on our post request that sends to the reliable company and unreliable company. If it returns a 409 error, which means that some seat in the booking is booked already by others. Then we will rollback by sending deleting ticket request to resume the availability of seats. In level 2, since we have local company, considering there are two steps including generating new ticket and store it, and set availability of seat to false, we use transaction to include the check of availability of seats to see if it is booked already together with two writing operation steps. We confirm order if seat is not taken or fail the transaction, and then rollback all the seats booking by deleting the ticket and resume the availability of seats in the same booking.

Question 15 We use sendGrid API to do the feedback. If a booking is successful, the program will invoke the sendEmail method, an email containing information as below will be sent: Dear starkkyrie@gmail.com, Below are your bookings: Mamma Mia Stalls:C9 On the other hand, if a booking fails, sendEmail method will be invoked to send a message as below before rolling back: Dear starkkyrie@gmail.com, Below are your bookings: Frozen the Musical Stalls:A4 Frozen the Musical Stalls:A4

Question 16 We do not make change to the configuration file since we consider that the current setting is more than enough than our needs. The default setting specifies maximum concurrent request to be 50, which in our test case cannot be reached. And the CPU usage ratio is set to 90 percent and the pending time for handle request is set to 15000ms in maximum and 5000ms in minimum. This figure do not need to be changed since the request we make to test the application is in a relatively small amount.

Question 17 Running a service or app natively usually requires users to install software or apps on machine. The service need to be designed suitable to the platform or operation system that it runs on. When using a web service, the users can get access to service despites of the usage of different operation system. The deployment on cloud enables the application starts to run automatically without manually start to run the application, which is the case of running in your local server.

Question 18 The local environment is different from the cloud platform. We encounter a lot of bugs and malfunctions because of the use of multiple threads on Cloud platform. And the program cannot find local files such as data.json and credential json files. The debug process becomes more difficult. When we deployed the program to cloud, we came across a lot of bugs which we never encountered locally. The log shows all the standard printout and exceptions in detail, but we have to do every attempt and re-deploy again, which costs a lot of time and energy.

Question 19 We use the public key provided by Google to verify the token signature. We use the google FireStore database to store the data model, google file store to store some files for program to read, and use pubsub as well. If we want to change to other cloud provider, the token signature verification process must be changed. While the pubsub, firestore and file store can be accessed as usual if we want to keep using them.

Question 20 A customer is estimated to send 10 to 20 requests to server from login to booking. Monthly on average the cost is 450,000 requests to a server. The scale will be billion level if we have 1 million customers per day. Currently, every request will be processed through database or connection to the remote theatre company. When the user amount rapidly increases, we consider that there are two perspectives for us to reduce the cost. The first is from customer's view. We can rely on caching on the chromes to prevent sending the same request, which will get the same result especially in terms of loading static resource file. From the server's view, we can return some static resources directly from Nginx. We can improve our code in model and database access to reduce unnecessary cost. We can also change the service logic to achieve asynchronous within the program.