Report: Software Architecture of Traffic Recording System

The SQL server performance drops significantly when the limit of TPS (transactions per second) is exceeded. Since our customers have thousands of data collectors recording traffic on roads globally, the system will experience a high volume of TPS, requiring additional servers to balance the load.

Assume the world is divided into regions. In each region, the TPS during peak time will not exceed the system's limit. Then we place servers in different regions. New tables are required to record information about the servers and regions, which helps identify the server and region that the customers need to access.

Table: servers

Field	Type	Description
serverid	INTEGER PRIMARY KEY	A unique, non-zero integer ID assigned to each server.
regionid	INTEGER	The region ID for where the server is located.
type	TEXT NOT NULL	Indicates the type of the server. It can be 'read', 'write, 'backup', or 'API'.
content	TEXT NOT NULL	Indicates the content of the server. It should be one of the 4 existing tables: locations, session, user, traffic, or runs server.py.

Table: regions

Field	Type	Description
regionid	INTEGER PRIMARY KEY	A unique, non-zero integer ID assigned to each region.
name	TEXT NOT NULL	A readable name of the region.

The locations table will need a new column with the corresponding region ID for each location.

For replication, we use the Global ID Transaction method, where each transaction has a unique ID. This transaction-based method simplifies the process of detecting the consistency between the sources and replicas.

To ensure data integrity, we need a read-only backup server for every database. This way, queries written to the original dataset will not affect the backup, as well

as in the event of a system fault, hardware failure, or other issues of the original database.

Table: users

Assume our customers travel between regions. They should be able to log in from anywhere. The following commands will be applied to this table:

Command	Style	Compute demand
Login	Read	High during peak time
Update password/Username	Write	Low
Backup	Read	Low

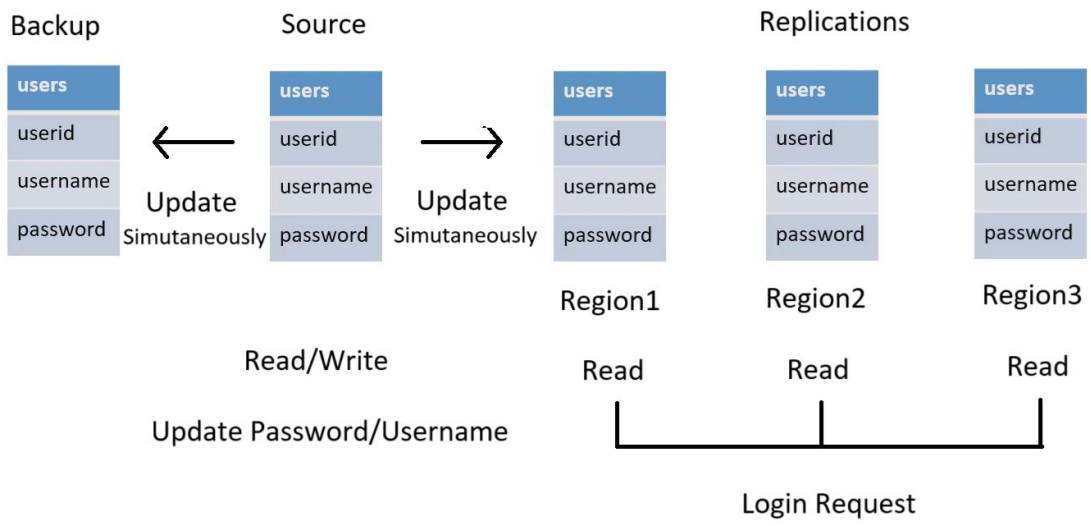

Figure1: Replication of users Table

For each region, we replicate the users table to handle high computing demand for login requests. Since users typically do not update their passwords frequently, these update requests do not require high computing resources and can be written directly to the source server. Replications and backups must be updated simultaneously to ensure users can log in after updating their information.

The locations, servers, and regions tables work similarly, with horizontal sharding according to the region ID in each region instead of replications.

Tables: session, traffic:

The demand for writing is high. Still, we need horizontal sharding according to the region ID instead of replications.

We require the following server configurations in each region:

Server	Content	Purpose	Time Zone	Demand
Read	Traffic/session records in total. Update monthly. Read only.	For data scientists to access, download, and analyse data.	Global	Storage: High Compute: High when survey
Write	Traffic/session records in the current month. Read and write.	For data collectors to record the traffic.	Local	Storage: Low Compute: High at peak time
Backup	Backup of the traffic record. Update monthly. Read only.	For database backup and integrity.	Global	Storage: High Compute: Low

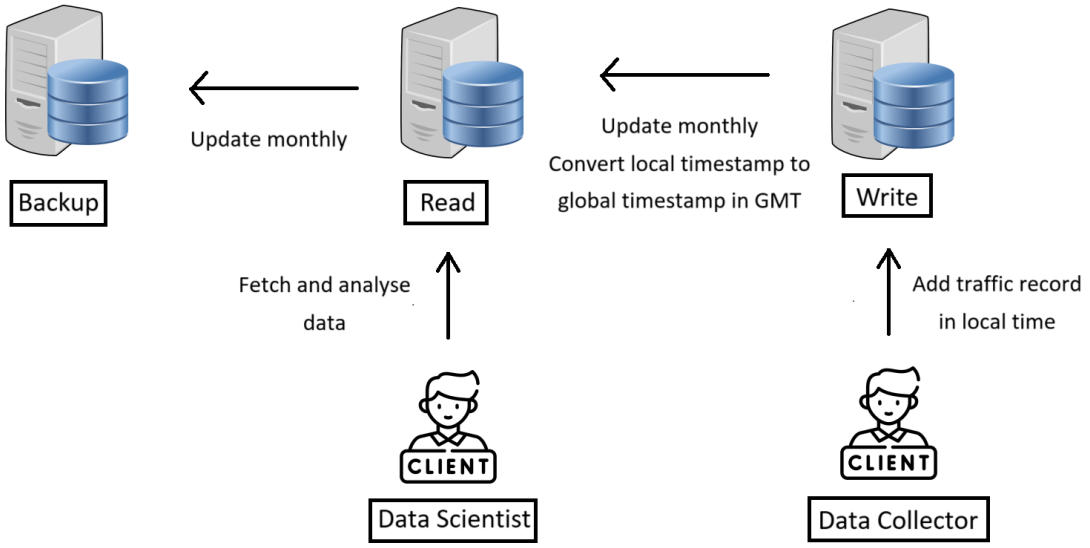

Figure2: Layout of servers for Table: traffic, session

When data collectors add or undo traffic records, the timestamp should reflect their local time zone. We can add a column to record the time zone in regions table if necessary. At the end of each month (or as needed based on customer requirements), the transactions are updated to the Read server, with the timestamp converted to GMT. This ensures a consistent GMT timestamp across all records, simplifying further analysis and preventing errors that may arise from time zone conversion.

For traffic record systems, the high computing demands only occur during peak times, such as weekday rush hours, or when the customer decides to conduct a survey. To minimise computing costs, one option is renting cloud services during

peak time, while another involves building servers with more threads, cores, CPUs, and RAM to improve the overall computing power and rent out extra computing resources to other companies during off-peak time. The main benefit of using cloud services is to reduce the risk of machine failure, as the service provider can quickly assign a replacement machine. In contrast, if servers are owned outright, a failure can be problematic due to limited hardware availability.

Furthermore, since the customers operate across different time zones, peak times from the two regions will not occur simultaneously. We can allocate computing resources across regions through the virtualisation layer: putting the majority in Region A during its peak and switching to Region B when its peak occurs. However, this introduces significant latency due to the physical distance between regions, which is constrained by the speed of light. Customers opting for this method should balance latency with computing costs.

For data collectors recording traffic, we provide pre-built images containing all the necessary tools and resources. For users conducting a survey, we offer containers that allow them to capture all the data in a volume and shut down the container once capturing is finished. We can lock the database on the Read server that is being analysed. This action will not disrupt the functionality of other servers. For example, the Write server can pause updates to the Read server until the survey is done.

For security reasons, server codes should run on a separate machine. User requests will route through the server code rather than directly accessing the database, which will be protected behind a firewall. Usernames and passwords must be encrypted and stored as hashes. Customers should not have access to other customers' databases, whereas their staff should be able to access their databases. This can be achieved by using firewalls between each database and granting access to the appropriate users. Additionally, virtualisation or containerisation layers can enhance security as they separate systems and applications. Customers will only have control of their virtual machines and cannot access the resources of others.

To add new features, two teams are required: one to write the code and another to write tests. If the feature is complex, it should be broken down into smaller functions. Unit tests, either white box or black box, should be written for each function, followed by integration tests for the entire feature. When a bug is fixed, a corresponding test should be created, and all regression tests should be rerun to ensure the new feature does not break existing functionality. Version control

should be used to track modifications. If the program's running time is too long, optimisation may be necessary. depending on whether it is one-time or repeated use.

When the customer requests the new feature to be deployed across all regions, it should not be applied to all servers simultaneously. Instead, to avoid the risk of a global failure due to unexpected errors, we will first update a single server. If it operates smoothly, we will gradually roll out the feature to other servers. Additionally, We should offer the old version of the application to the customers who prefer not to use the new feature.