## Brief design review:

Application is written using dropwizard. I've never worked with it before and it seems much simplier and lighter than other frameworks I've worked with.

App consist of one module though it'd be better to separate it in 2-3 different ones but I separated sources by package structure for simplicity.

As standart DW app it has Configuration(TrueAppConfiguration) and Application (TrueApplication) classes. I've used one single Resource class as task seems pretty small and it looks ok in one Resource boundaries. I have only 2 working methods (3 actually, but 3rd is just for testing so you are able to review archived items) - "get user" and "get user statistics".

To run app:

1. Check-out Git project from "https://github.com/simpleAndrew/true"
2. run mvn clean package
3. execute run.sh from root folder of project (if you're using windows, just copy the content of run.sh and execute it)

After it's started, you can access app on localhost:8080 (hopefully you haven't had anything running on this port):

- localhost:8080/users/1?userId=0 - will show you 1st user
- localhost:8080/users/1/read-statistics?userId=0 - will you read ocurences

You can play with created users only (see yaml config for users) - ID is autoincremented so you can user numbers from 1 to 5.

Speaking about DB. It's very simple in my case. I use only 2 tables:

- **table true_user(id long auto_increment primary key, name varchar(100))**
- **table true_user_statistics(id long auto_increment primary key, viewer_id long, requested_user_id long, read_date timestamp)"**

First to keep users, second to keep read calls. User table is single instance, inited with own DataSource - it allows to keep users separate from Statistics tables. It has just primary key index on ID as usual.

Statistics tables are sharded. To make sure we can keep good speed with million of requests, application might be started with multiple separate DataSources. All user read calls will be distributed evenly based on user id. It's simple algo, but while I have no other information

about read times for different users, it seems ok. These statistics tables are indexed by read_date and requested_user_id to be able to find required data quickly.

Also to make sure indexes are working not very slow, I've implemented batch job which will remove redundant data from statistics tables and keep it in separate datasource as in archive. This job is configured and executed based on number of redundant records found in sharded statistics table. Number of records, delays between execution are configured.

So, answering on questions from initial spec:

1. I do delete it from one tables and copy to separate storage.
2. Yes, I have. It copies data into archive.
3. I think I spent more resources on managing datasources to maintain application quick response.

Speaking about patterns, I've used Proxy (UserStatisticsProxyDao) and Strategy (DaoProvider) to implement shard management on app level. Also AbstractFactory might be recognised in *ToolsFactory classes, but they're more auxiliary ones to keep code readable.

To test application I'd recommend to check-out fast_dump branch as I've changed properties there to run dump job every 10 seconds and copy all records from real tables when ther're more than 4 of them in shard. You'll be able to see the dynamic or archive tasks. To read data from archive just use:

- localhost:8080/users/read-statistics/dump?userId=0

That's pretty much it. Thank you for your time and interesting task that exercise my brain a little bet)

Best Regards,
Andrew Shchyolok