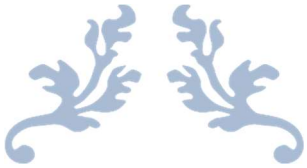

DISTRIBUTED DATABASE PROJECT FINAL REPORT

By: XueYing Hu, Zeqing Li

JUNE 17, 2018

Abstract

This project aims to build an efficient and effective system to support an application that allow user to read interesting and destressing articles. The system composes of two layers: client interface and data center. A command line user interface is implemented to demonstrate possible user operations and interaction with the data. The data center consists of two sets of Redis cache and MongoDB pairs running representing different locations running in Docker containers and HDFS running in a virtual machine. Design and implementation considerations revolves around meeting user requirements and optimizing efficiency of queries.

Problem Background and Motivation

With the development of social economy, more attention is being paid to mental health and the pursuit of the spiritual world is increasing. Students are increasingly becoming more stressed in this fast-paced world. There needs to be more resources for them to read articles that relate to their interests and help them relieve the burden.

This project will set up an electronic management system to provide users with a better reading and destressing experience. This system is designed to provide a smooth user experience and support various user operations. First, users can query for articles they want to read. Second, user can create new articles. Third, the system needs to show popular articles to help users find new articles they might be interested in. As the project is just starting, the system aims to be flexible and efficient.

Existing solutions

Traditional server-side architecture has been shaken up by new technologies that better streamline infrastructure and maximize server resources (Upwork, n.d.). Containerized applications for deployment in cloud platforms have become increasingly popular. Docker has been the driving force behind this trend, where 2/3 of companies have tried to use it (Novoseltseva, 2017). It ensures consistency and standardization across platforms and allows for faster configuration. This is more light-weight and easy to use compared to virtual machines. Hadoop does not have an official Docker image and thus virtual machine is used instead.

There are popular choices in the industry with regards to data storage. For relational databases, common existing solutions for server-side database include MySQL, Oracle, SQL Server, etc. These database share similar properties such as standard storage in tables, existence of primary and foreign keys to establish relationship, similar syntax to operate, and support for scalability and high-performance (Upwork, n.d.). Oracle and SQL Server are not free, while MySQL is open source. Each company has additional support for their own database, for example, .NET framework can be easier to integrate with SQL Server. On the other hand, MongoDB is representative of NoSQL databases, where it has collections and JSON-like documents instead of tables and rows. There is more flexibility in changing the structure of the documents and having a hierarchical structure by using subdocuments. The current application is still in start-up phase, thus flexibility and low cost is important. Therefore, MongoDB's properties are more suitable for this use case providing superior scalability and performance (mongoDB, n.d.).

A cache provides faster querying speed especially when dealing with a lot of users performing similar operations at the same time. Two popular options are Redis and Memcached, which are both in-memory key-value data stores (Vimal, 2017). They both support super fast caches stored in RAM, however, Redis contains more powerful features. Memcached is better used for small and static data without support for different data types. On the other hand, Redis is more of a "data structure server" and provides greater power and efficiency (Vimal, 2017). Since data from the database contain more complex structures and relations, Redis is the more preferred option.

Solution Design and Implementation

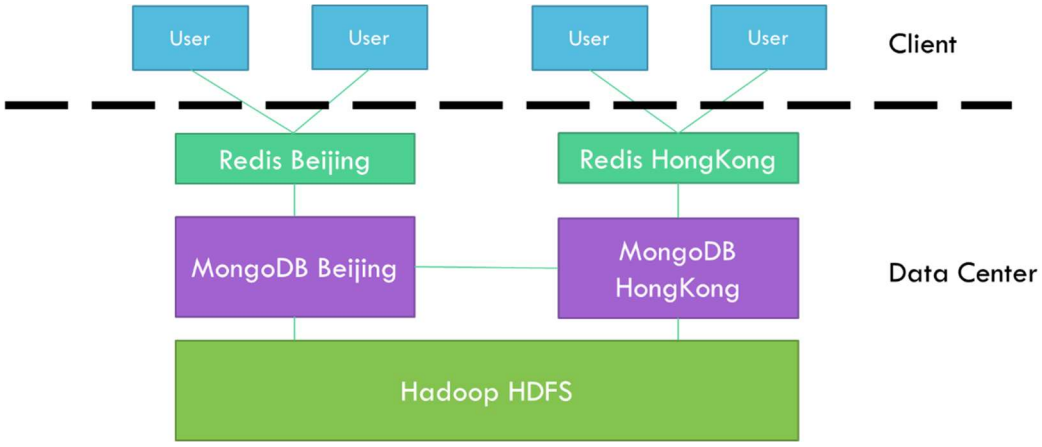

Figure 1 Architecture of the system

To support the goals of the application, the solution architecture separates the client operations from the data center or server-side operations. Depending on the user location, user inquiries will either first go to the Redis cache in Beijing or Hong Kong. If the data is not in cache, then the query will reach the corresponding MongoDB and if it is still not found, then the system will try to find it in the database at the other location. Hadoop HDFS is used to store unstructured data, for example, article text, image and video.

MongoDB

The different table are stored as collections in MongoDB and each data point is a document. User table is fragmented by region, hence the separate Beijing and Hong Kong database. The Read table is fragmented based on the User table by "uid". Article category is used to fragment the Article table, where MongoDB Beijing holds only articles with category "science" while MongoDB Hong Kong holds category "science" and "technology". Be-Read table is fragmented based on the Article table using "aid" and the Popular Rank table is replicated in both databases.

MongoDB is supported by Docker containers. The MongoDB Docker images is based on the official MongoDB Docker image. It is pre-loaded with all the data based on the fragmentation, there is a separate Docker image for Beijing and Hong Kong. Each has a Docker volume attached, which persists data from the container to the host machine. This is a common solution to prevent data being lost after container shut down.

We added some minor changes to the data model. First of all, "uid" lists in the Be-Read table have been changed to an array of subdocuments that holds "uid" and the "timestamp" of Be-Read. Since the Be-Read table did not differentiate the time of read, comment, share, and agree, the timestamp is assumed to be the same of all actions. This makes calculating the time dependent Popular Rank table easier. The Popular Rank table is calculated based on the top 5 total number of reads during the specific time range. Timestamp in Popular Rank stores the beginning the calculated time period, for example, it was weekly then timestamp is 0:00 on the first day of the week – Monday. In Popular Rank

table, the article “aid” list is also stored as an array of subdocuments containing the “aid” and the total number of read. This way it’s easy to track the exact rank of the Article from first place to fifth.

Another consideration for the database schema is adding indexes (Figure 2). User has 2 unique indexes, “uid” and “name”. “name” is added because the system uses user’s name to login. Read ideally would have a unique key that combines “aid” and “uid” as users should have one joined record per article. The other indexes are to speed-up the calculation for Be-Read table. The complete list is shown below.

User	Article	Read	Be_Read	Popular_Rank
uid (U) name (U)	aid (U)	aid uid aid, uid (U) aid, ReadOrNot aid, agreeOrNot aid, shareOrNot aid, commentOrNot	Aid (U) readNum	temporalGranularity, timestamp (U)

Figure 2 Indexes in MongoDB for each collection. (U) indicates unique index.

Redis Cache

Data retrieved from the database are stored in a Redis cache. This require remodelling collections and documents as key-value pairs. The decision on what to use as key depends largely on which search queries are the most frequent. Complex queries that do not leverage the value of the key will require running $O(n)$ through all the records in Redis. This is not efficient and greatly complicates the code, thus these more rare and complex queries hit the database directly. For the User table, searching by “uid” and “name” is the most common, so from user’s name the “uid” can be obtained, and the full data is stored in the User “uid” key. For Read, since a user is already logged in, the query would already know the “uid” that maps to a list of “aid”, then each Read record for a particular “aid” can be obtained. The other key-value pairs are shown in Figure 3.

User	Article	Read	Popular_Rank
Key: “USER”+name Value: uid string Key: “USER”+uid Value: hashmap	Key: aid Value: hashmap	Key: “READ”+uid Value: aid list Key: “READ”+uid+aid Value: hashmap	Key: “Popular_Rank” +timestamp + temporalGranularity Value: hashmap

Figure 3 Key value pairs in Redis cache for each table. Hashmap is the dictionary object that MongoDB returns for a document.

Docker is used to support Redis as well. Redis cache’s eviction policy is set to be “allkeys-lru”, meaning that all keys are can be removed based on the Least Recently Used principle. Maximum memory is set at 100MB. When a modify occurs, the database is modified first, if successful, then the record in the cache is modified. This guarantees the integrity of the

data in the database. Data in the cache is not persisted, so Docker volumes are not needed in this case.

Hadoop

The Hadoop layer is used to store unstructured data such as text, pictures and videos. Based on the structure of the system, the Hadoop layer is the lowest level, it only needs to receive upper level parameters and instructions, and provides corresponding services for the upper level. Specifically to this system, the Hadoop layer needs to receive two parameters, the first parameter is read/write, which is used to indicate whether to read or write data from the Hadoop layer, and the second parameter is aid, which is used to specify the ID of the operating data.

User Command Line Interface

A series of operation is supported by the command line interface to interact with the data mentioned above. There are two user types, one is admin and the other are normal users. The admin user can operate on all users, for example, querying and deleting users. The admin also has the right to run calculations for Be-Read and Popular Rank table. Since the Popular Rank table produces a new entry daily, there's no need to continuous update these tables, but rather the admin runs the process at the end of each day. The normal user can create a new account or update an existing account.

After user logged in, they can query articles to read, commend, share, and agree which will update the Read table as needed. They can also create new articles by providing the information and the folder that contains the text, image, and video. Popular Rank table can be queried by date and the temporal granularity, which will display the top 5 list of articles for the given criteria. The user can also check "history", which is the Read table and see what they have read.

Solution Evaluation

Using Docker containers to support most of the infrastructure for the project is very effective solution. Deployment and replication has been greatly simplified and the system can be easily setup on a different machine. Docker also has support to monitor the status of the running containers.

MongoDB has been quick to pick up for someone with knowledge of relational databases. Allowing subdocuments in the data schema is an advantage over traditional tables as there's greater flexibility in the structure of the data. Furthermore, complex queries and data aggregation are well-supported in MongoDB. Indexes made a big difference when computing the Be-Read and Popular Rank table, reducing the calculation time from a couple of hours to around one hour.

The combination of MongoDB and Redis have in general sped up the system. Figure 4 shows statistics in the length of time it took to complete querying operations in MongoDB and in Redis. There is significant decrease in the time it takes when using cache compared to MongoDB when the search method matches the way the key-value pair is stored in cache. It is notable when querying by conditions that are not stored as key, it takes linear time to

look through all existing records in the cache, which takes longer than retrieving it form MongoDB.

	Time Elapsed Retrieving Data from MongoDB (µs)	Time Elapsed Retrieving Data from Redis Cache (µs)
Querying Article by aid	155040	5475
Querying Read without conditions	308153	181663
Querying Read with Conditions (shareOrNot=1)	272685	356455
Querying User without conditions show first 10	539147	111172

Figure 4 Time elapsed to retrieve data from MongoDB and Redis.

In this project, Hadoop is only used to store three tuples of video documents and pictures. It does not make use of Hadoop to do computation. In fact, it wastes a lot of advantages of Hadoop. In future work, we can make full use of the programming model of map-reduce in Hadoop, run some large data algorithms on Hadoop, such as collaborative filtering recommendation algorithm, on the one hand, better use of Hadoop, on the other hand, provide more intelligent service for users and enrich user experience (Wilmos, 2017). In addition, the project is only a demo, the number of nodes of the Hadoop is far from the actual demand, so there is no consideration of many scheduling issues. When the node is extended to a certain number, even using only Hadoop for storage, we must also design a good scheduling strategy, which can be followed up in the future (Holmes, n.d.).

Conclusion

The data center setup with the combination of Redis, MongoDB, and Hadoop has effectively met the goals of the project. It provides storage for structure and unstructured data as well as flexibility to continuously change and expand the system. The containerized nature of MongoDB can easily support future additions of replicas or standby databases. Integration with Hadoop can also be expanded to make use of other features as needed. Furthermore, querying speed for data has been optimized for users with the Redis cache and with more knowledge of user behaviour, better keys can be designed to meet user needs. In conclusion, this project successfully developed a preliminary system for the application's server-side operations.

Work Load Allocation

XueYing Hu completed the user command line interface, Redis cache, and MongoDB implementation and infrastructure setup.
Zeqing Li completed the Hadoop implementation and setup.

References

DellEMC. (n.d.). *Modern Data Center*. Retrieved from DellEMC: https://www.emc.com/corporate/glossary/modern-data-center.htm

Holmes, A. (n.d.). *Source code for book "Hadoop in Practice", Manning Publishing*. Retrieved from Github: https://github.com/alexholmes/hadoop-book

mongoDB. (n.d.). *Advantages Of NoSQL*. Retrieved from mongoDB: https://www.mongodb.com/scale/advantages-of-nosql

Novoseltseva, E. (2017, April 27). *Top 10 Benefits of Docker*. Retrieved from DZone: https://dzone.com/articles/top-10-benefits-of-using-docker

Upwork. (n.d.). *MS SQL vs. MySQL: Which Relational Database is Right for You?* Retrieved from Upwork: https://www.upwork.com/hiring/data/sql-vs-mysql-which-relational-database-is-right-for-you/

Upwork. (n.d.). *What Is Containerization and Is It The Right Solution For You?* Retrieved from Upwork: https://www.upwork.com/hiring/development/what-is-containerization-and-is-it-the-right-solution-for-you/

Vimal, R. (2017, May 4). *Memcached vs Redis, Which One to Pick?* Retrieved from LinkedIn: https://www.linkedin.com/pulse/memcached-vs-redis-which-one-pick-ranjeet-vimal/

Wilmos, F. (2017, August 7). *Hadoop 在 Centos7 下的单机布署(一). Standalone Operation*. Retrieved from 镜缘浮影: http://soft.dog/2017/08/07/hadoop-01-standalone/

System Manual

System Requirements

- Ubuntu 16.04 LTS or Ubuntu 18.04 LTS
- Java 1.8.0_171
- Hadoop 2.9.1
- Python 3.6.2
    - Libraries: pymongo, redis, tabulate
- Docker 18.03.1-ce

MongoDB Setup Instructions

The Docker images for MongoDB needs to be created.
In folder db_bj, run:

```
1. docker build -t mongo_bj .
```

In folder db_hk, run:

```
1. docker build -t mongo_hk .
```

Create the Docker volumes for both MongoDB containers:

```
1. docker volume create db_hk
2. docker volume create db_bj
```

Start the MongoDB containers:

```
1. docker run -d -p 27016:27017 -v db_hk:/data/db2  mongo_hk
2. docker run -d -p 27017:27017 -v db_bj:/data/db2  mongo_bj
```

Note: The ports correspond to those specified in cl_ui/config.json.

Redis Setup Instructions

Build the Docker image for Redis cache:

```
1. docker built -t redis_lru .
```

Start the two Redis container:

```
1. docker run -d -p 6379:6379 redis_lru
2. docker run -d -p 6380:6379 redis_lru
```

Note: The ports correspond to those specified in cl_ui/config.json.

Hadoop Setup Instructions

First, open the terminal window and enter the following commands to create a new user:

```
1. sudo useradd -m hadoop -s /bin/bash
```

Then use the following command to set the password, which can be set to Hadoop simply.

```
1. sudo passwd Hadoop
```

Adding administrator privileges for Hadoop users：

```
1. sudo adduser hadoop sudo
```

Finally, log off the current user and login with the newly created Hadoop user on the login interface.

After logging in with Hadoop users, we first update the apt:

```
1. sudo apt-get update
```

Cluster mode requires SSH login, Ubuntu has installed SSH client by default, and SSH server is also required.

```
1. sudo apt-get install openssh-server
```

After installation, you can log on to this machine with the following commands:

```
1. ssh localhost
```

But landing in this way needs to input the password every time. We need to configure it as SSH without password.

The first exit just SSH, we went back to the original terminal window, and then use ssh-keygen to generate keys, and the key to authorization:

```
1. exit
2. cd ~/.ssh/
3. ssh-keygen -t rsa
4. cat ./id_rsa.pub >> ./authorized_keys
```

Download Hadoop and install it into /usr/local/:

```
1. sudo tar -zxf ~/Download/hadoop-2.9.1.tar.gz -C /usr/local
2. cd /usr/local/
3. sudo mv ./hadoop-2.9.1/ ./hadoop
4. sudo chown -R hadoop ./hadoop
```

Hadoop can be used after unzip. Enter the following command to check if Hadoop is available. Success will display the Hadoop version information:

```
1. cd /usr/local/hadoop
2. ./bin/hadoop version
```

Hadoop can run on a single node in a pseudo - distributed manner, the Hadoop process runs in a separate Java process, the node is both as a NameNode and as a DataNode, while reading a file in the HDFS.

The configuration file of Hadoop is located in /usr/local/hadoop/etc/hadoop/, and pseudo distribution needs to modify 2 configuration files core-site.xml and hdfs-site.xml. The configuration file of Hadoop is XML format, and each configuration is implemented by declaring the name and value of property.

Modify the configuration file core-site.xml， change

```
<configuration>
</configuration>
```

to the following configuration:

```
<configuration>
        <property>
             <name>hadoop.tmp.dir</name>
             <value>file:/usr/local/hadoop/tmp</value>
             <description>Abase for other temporary directories.</description>
        </property>
        <property>
             <name>fs.defaultFS</name>
             <value>hdfs://localhost:9000</value>
        </property>
</configuration>
```

Similarly, modify the configuration file hdfs-site.xml:

```
<configuration>
        <property>
             <name>dfs.replication</name>
             <value>1</value>
        </property>
        <property>
             <name>dfs.namenode.name.dir</name>
             <value>file:/usr/local/hadoop/tmp/dfs/name</value>
        </property>
        <property>
             <name>dfs.datanode.data.dir</name>
             <value>file:/usr/local/hadoop/tmp/dfs/data</value>
        </property>
</configuration>
```

After the configuration is completed, formatted NameNode

```
1.  ./bin/hdfs namenode -format
```

Then open the NameNode and DataNode daemons.

```
1.  ./sbin/start-dfs.sh
```

The following WARN hints may appear at startup: WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... Using builtin-java classes where applicable. The WARN hint can be ignored and will not affect normal use. After the boot is completed, you can tell if you have successfully started by calling JPS. If successful, the following processes will be listed: "NameNode", "DataNode" and "SecondaryNameNode".

To use HDFS, you first need to create user directories in HDFS:

```
1. ./bin/hdfs dfs -mkdir -p /user/hadoop
```

Then the data_hdfs is created to store the data:

```
1. ./bin/hdfs dfs -mkdir data_hdfs
```

To close Hadoop, run:：

```
1. ./sbin/stop-dfs.sh
```

Hadoop Example

Suppose that the data in the current Hadoop is empty, we need to write data of aid 1 into HDFS, and the system calls the instructions to upload the local data to the HDFS.

```
hadoop@ubuntu:/usr/local/hadoop$ ./bin/hdfs dfs -put ./data_local/1 data_hdfs
```

```
hadoop@ubuntu:/usr/local/hadoop$ ./bin/hdfs dfs -ls data_hdfs
Found 1 items
drwxr-xr-x   - hadoop supergroup          0 2018-06-16 23:36 data_hdfs/1
```

When we read data from aid to 1 from HDFS, we can download data from HDFS to the local data_read_aid folder by instruction, in this case, data_read_1, as shown below:

```
hadoop@ubuntu:/usr/local/hadoop/cl_ui$ ../bin/hdfs dfs -get data_hdfs/1 ./data_read_1
```

```
hadoop@ubuntu:/usr/local/hadoop/cl_ui/data_read_1$ ls
1.Flv  img1.png  text1.pdf
```

The data is a video text picture three tuple.

Command Line UI

To run the command line user interface, all above setups must have been completed with 4 Docker containers running and HDFS started.
In folder cl_ui:

```
1. python3 start_ui.py
```

You can also run in test_mode, which will output if the data was retrieved from cache, using:

```
1. python3 start_ui.py test
```

The first command will ask whether you are a returning user.

- *Y* – will ask for a username
    - *admin* – current username to access admin operations, defined in cl_ui/config.json.
- *N* – will prompt to enter to user data

Admin user

- *goto admin_ops* – only allowed operation
    - *query user*

    - *delete user*
    - *compute*
        - Computes Be-Read and Popular Rank tables
        - This must be run first at system creation, afterwards, it will be saved in the Docker volume

Normal user

- *goto article*
    - *query*
        - Can query article by id or by article property
        - Then user has the option to read, comment, share, and agree on the article by aid
    - *create* – enter data for create a new article
- *goto popular*
    - Will prompt for the date you wanted for and the temporal granularity
    - Then user has the option to read, comment, share, and agree on the article by aid
- *goto user*
    - modify – modify user data
    - history – query Read table

Note: All query requires that the property name match the name stored in the database. If show_count is an option when querying, it specifies how many entries to show at a time and by default it is 1.