Solution for Trusting social Exercises

Tran Tuan Manh

November 24, 2017

Contents

1 Task 1 **1**
1.1 Algorithm 1
1.1.1 Strategy 1
1.1.2 Pseudo code 2
1.2 Complexity 3
1.2.1 Time complexity 3
1.2.2 Memory complexity 3

2 Task 2 **3**
2.1 High level architecture 3
2.2 Infrastructure 4
2.3 Ingestion 4
2.4 Storage 4
2.5 Processing 4

1 Task 1

1.1 Algorithm

1.1.1 Strategy

We will reduce records by phone number. Then, each phone number have a list of activation date and a list of deactivation date. Records see bellow:

```
Record 1:
PHONE_NUMBER:       0987000001
ACTIVATION_DATE:    [2016-03-01,2016-01-01,2016-12-01,2016-09-01,2016-06-01]
DEACTIVATION_DATE:  [2016-05-01,2016-03-01,2016-12-01,2016-09-01]

Record 2:
PHONE_NUMBER:       0987000002
ACTIVATION_DATE:    [2016-02-01,2016-03-01,2016-05-01]
DEACTIVATION_DATE:  [2016-03-01,2016-05-01]

Record 3:
PHONE_NUMBER:       0987000003
```

```
ACTIVATION_DATE:    [2016-01-01]
DEACTIVATION_DATE:  [2016-01-10]
```

With each phone number, we have:

```
len(ACTIVATION_DATE) = len(DEACTIVATION_DATE)
or
len(ACTIVATION_DATE) = len(DEACTIVATION_DATE) + 1
```

Then, we will ascend sort list of activation date and list of deactivation date. And we iterate on list of activation date and list of deactivation date. If activation date equals deactivation date then we mark activation date to remove. Actual activation date is the last element of remain activation date list.

1.1.2 Pseudo code

In this section, we will present find actual activation date algorithm. Reduce by phone number algorithm and sort algorithm are not presented.

Algorithm 1: Find actual activation date algorithm

Function *Find_actual_activation_date(activation_date, deactivation_date)*
- **Result:** *actual_activation_date*
- *actual_activation_date_array* := *activation_date*;
- $i := 0$;
- $j := 0$;
- **while** $i <$ *length(activation_date) and* $j <$ *deactivation_date* **do**
  - **if** *activation_date[i] = deactivation_date[j]* **then**
    - *actual_activation_date_array*$[i] := -1$;
    - $i := i + 1$;
    - $j := j + 1$;
    - break;
  - **end**
  - **if** *activation_date[i] > deactivation_date[j]* **then**
    - $j := j + 1$;
    - break;
  - **end**
  - **if** *activation_date[i] < deactivation_date[j]* **then**
    - $i := i + 1$;
    - break;
  - **end**
- **end**
- $k :=$ *length*(*actual_activation_date_array*);
- **while** $k > 0$ *and actual_activation_date_array[k] = -1* **do**
  - $k := k - 1$
- **end**
- *actual_activation_date* = *actual_activation_date_array*$[k]$
- **return** *actual_activation_date*

1.2 Complexity

1.2.1 Time complexity

Our algorithm includes 3 partitions: reduce by phone number, sort activation date and deactivation date, find actual activation date. Time complexity of partition follows:

- **Reduce by phone number**: $O(n)$, with n is total records (500,000,000)
- **Sort activation and deactivation**: $O(m \times log(m))$, with m is number of phone number's activation/deactivation date
- **Find actual activation date**: $O(m)$, with m is number of phone number's activation/deactivation date

Time complexity:
$O(n) + k \times (O(m \times log(m)) + O(m)) = O(n) + k \times O(m \times log(m))$ with k is number of phone number

1.2.2 Memory complexity

Algorithm do not execute all data in once time. Each task will load k phone number to process. So, space complexity is:
$O(k) \times O(m) = O(k \times m)$, with k is number of processed phone number, and m is size of activation/deactivation date.

2 Task 2

2.1 High level architecture

We can break the high-level design of our example into five main sections: storage, ingestion, processing, analyzing and orchestration. Figure 1 depicts an overview of such a design.

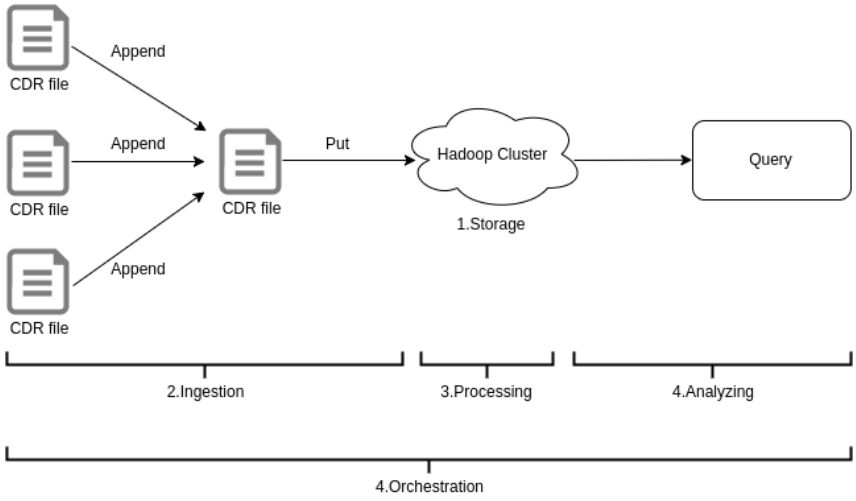

Figure 1: High level architecture of the system

In our design implementation, we will use HDFS to storage data. About ingestion, because we have CDR files, so we append CDR files to the larger CDR files. The appended

CDR files will put to HDFS. We will use Spark for processing and Impala for analyzing the processed data.

2.2 Infrastructure

Suppose, our cluster has 15 servers. Hardware specifications follows Table 1. We have 3 servers for Zookeeper cluster, 1 server for NameNode, 1 server for Second NameNode and 12 servers for DataNode. We will install Spark Worker Node on 12 DataNode servers.

2.3 Ingestion

We have 70,000 CDR files and total file's size is about 200GB. So, file size is 2.9MB. It is smaller than default size of block(64MB) on HDFS. Note that, HDFS works very good with a large file and very bad with many small files. So, we must to append CDR files to a larger CDR file. Then we put CDR file into HDFS.

2.4 Storage

We use HDFS for storing data. Each data block is replicated on 3 DataNode. We use Parquet for storing the processed data that will be analyzed later on. General, processed data is smaller than raw data. Suppose, processed data is 100GB. Because we have 12 DataNode, each part will be at least 768MB in size, we will split processed data into about 120 files (use coalesce(120) before save to HDFS).

We still storing raw data after processed. Because ETL can failed and we want to reprocess raw data.

2.5 Processing

We use Spark and Spark Sql for processing data. Figure 2 describes the processing.

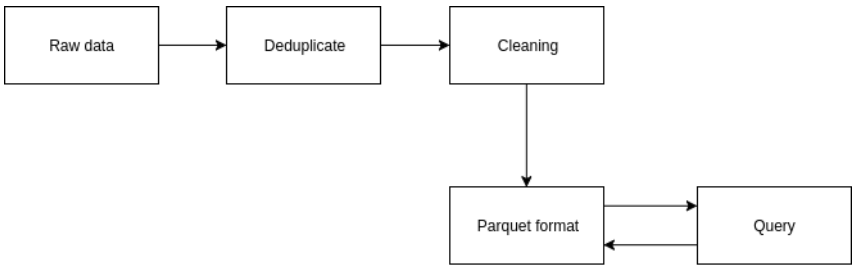

Figure 2: Processing design

Begin, extract the interesting data from raw data. Transform the extracted data to create processed(deduplicated, cleaned) data set. Finally, storing processed data set in HDFS.

We will query the data by accessing HDFS, then we will storing result in HDFS.

Server	Hardware	Processing	Notes
1	• **CPU**: 12 cores 2.5Gh • **Ram**: 128GB • **Storage**: 5 1TB hard disks, 1 for OS, 2 for FS (RAID 1), 1 for Zookeeper, 1 for Journay node	Zookeeper, NameNode, Journay Node, JobTracker	NameNode, JobTracker
2	• **CPU**: 12 cores 2.5Gh • **Ram**: 128GB • **Storage**: 5 1TB hard disks, 1 for OS, 2 for FS (RAID 1), 1 for Zookeeper, 1 for Journay node	Zookeeper, Second NameNode, Journay Node	Second NameNode, JobTracker
3	• **CPU**: 12 cores 2.5Gh • **Ram**: 128GB • **Storage**: 5 1TB hard disks, 1 for OS, 2 for FS (RAID 1), 1 for Zookeeper, 1 for Journay node	Zookeeper, Journay Node	
4	• **CPU**: 12 cores 2.5Gh • **Ram**: 256GB • **Storage**: 12 2TB hard disks	DataNode	DataNode, TaskTracker
5	like server 4	like server 4	like server 4
6	like server 4	like server 4	like server 4
7	like server 4	like server 4	like server 4
8	like server 4	like server 4	like server 4
9	like server 4	like server 4	like server 4
10	like server 4	like server 4	like server 4
11	like server 4	like server 4	like server 4
12	like server 4	like server 4	like server 4
13	like server 4	like server 4	like server 4
14	like server 4	like server 4	like server 4
15	like server 4	like server 4	like server 4

Table 1: Cluster