**YCBS-257 - Data at Scale – Khaled Tannir**

Final Exam

**30%** Sunday Mar 10 2019 – 11:59 PM

**STREAM DATA INGESTION AND ANALYSIS CASE STUDY**

**SCENARIO**

As a professional Data at Scale consultants, you have been asked to perform 2 mandates:

1. Prepare an assessment of a key project at **Big Company Group** (BCG) with your recommendations to answers some strategic questions for that project and proposed action to implement your solution;

and

2. Help the developers team in creating data ingestion and analysis scripts using Hadoop commons tools such as Pig, Hive, Impala, Sqoop and Flume to improve data ingestion and analysis practices.

**REQUIREMENT**

• This is an individual assignment and should reflect your understanding of the Hadoop ecosystem and the data at scale problems we are trying to solve.

• You are invited to not share this document in any format and for any reason.

**THE CASE STUDY**

• The Case Study is based on a real project environment that highlights a number of challenges often found in Hadoop project development. Many of these were addressed as specific topics within the course.

*As such you are encouraged, for each course topic, to identify any related information within the Case Study and to use the course lectures to gain insights for dealing with these challenges*.

• **Big Company Group** needs to evaluate the data stream collected in near real-time (< 1 sec ) in order to use this data in machine learning, predictive analysis, customer behavior, fraud detection and so on…

• **Big Company Group** develops IoT (Internet of Things) sensors. For a particular experience, there are **5,000** sensors equally split between two cities. There are multiple sensors each with their own data elements, but they follow a common data format. o Each sensor transmits on average **50 attribute** records every second.

* 1. o Each sensor’s data attribute row is on average **100 bytes** wide.

**PROJECT REQUIREMENTS**

• Ingest the sensor data into the HDFS store for evaluation and processing.

• Corporate data standards require all input sensor data to be persisted for **12 months**.

• All QA test results data (estimated to 10% of the ingested data per year) to be stored for a period of **5 years**.

**PLATFORM PLANNING (10%)**

The most common practice to size a Hadoop cluster is sizing the cluster based on the amount of storage required. The more data into the system, the more will be the machines required. Each time you add a new node to the cluster, you get more computing resources in addition to the new storage capacity.

To determine the memory size of the NameNode server, we need to **add the memory** needed by NameNode to manage the HDFS cluster metadata (in memory) and **the memory needed** for the Operating System.

The IT department defined the hardware configuration for each node in the cluster: **1 CPU 8 core**, **32 GB** memory and **20 TB** Hard Disk Drive. The data **node storage capacity** was calculated as **13 TB**

• Assuming a replication factor of **3** how many nodes in the cluster?

• Assuming a HDFS bloc size is **128 MB** and each block need **680 Bytes** for its metadata. What is the recommended NameNode memory size?

• What are the Systems architecture options for pulling the data from the source location in preparation for ingesting into HDFS?

***\*\*\*Conversions made assuming 1024 bytes in a kilobyte\*\*\****

***Calculate Data Storage Requirement:***

# Sensors = 5000

Record Rate = 50 records/s

Record Size = 100 bytes

***Corporate Requirement:***

Time Persisted = 12mo

Storage Requirement = (5000)\*(50 records/s)\*(100 bytes)\*(31,556,952 s/year) = 717.5220 TB

***QA Requirement:***

Time Persisted = 5 years

Fraction of Data = 0.1

Storage Requirement = Corporate Storage Requirement \* 0.1 \* 5 = 358.761 TB

Total Storage Requirement = 1076.28 TB

# Nodes Required = (Memory Required)\*(Replication Factor)/(Node Storage Capacity)

# Nodes Required = (1076.28 TB)\*(3) / (13 TB) = 248.37

# Nodes Required = 249 (rounding up to nearest integer)

HDFS Block Size = 128MB = 0.000122 TB

Total Storage Requirement = 1076.28 TB

HDFS Blocks = 1076.28 TB / 0.000122 TB = 8,821,968

NameNode Memory Size = (HDFS Blocks) \* (Block Size) \* (Replication Factor) = 8,821,968 \* 680 bytes \* 3 = 16.76 GB

Remarks on Data Ingestion:

In terms of data ingestion I would recommend installing a Flume agent that would sync the Flume source to a specified location (could be a given machine’s port) to collect the streaming data from these sensors. Since IoT companies usually deal with data in motion this seems to be a preferred option over a one-time dump system such as Sqoop. Alternatively Big Company Group could be set up with Spark Streaming or Storm to injest this real-time data.

**PLATFORM INSTALLATION (10%)**

• Describe the relative merits for installing and running the cluster in the cloud vs on-premises.

• How would you verify that the cluster can support the volumes in the project requirements?

Cloud vs On-Premisis:

It’s fairly clear that on-premises solutions of all types provide greater control and security. The con with this approach is that not every company can afford to deploy, maintain and expand on an on-premise framework. Cloud solutions has the advantage of mobility, accessibility and reliability (most cloud solutions comes with 99.9% uptime guarantee). They also have the advantage of scaling indefinitely and being centralized rather than the fragmented on-premises approach.

Verifying the Sizing:

To verify the cluster can support the volumes in the project requirements I would stress-test the system after the hardware was set up and the framework was deployed & installed. This would mean I would stream or dump the calculated theoretical memory requirement into the system to see if it responds well to this volume of data. If it does – we would delete all the dummy data and return it to its clean state for deployment - if it does not – we would need to go back to the drawing board and decide if we made a mistake in calculating the required setup or if it is a deployment issue.

**DATA INGESTION AND DATA MODELING (80%)**

• Given the above project requirements, which components would you use to ingest the data?

To ingest this data I would use a data-in-motion collector such as Flume, Spark Streaming or Storm to collect the data in HDFS for storage and analysis. My assumption here is the sensors are dumping their packets of data on the fly so there will need to be an agent setup to collect these packets to store in HDFS. To analyze the data once it’s in HDFS I would likely use Hive or Impala depending on the complexity of the query required. Pig could also be useful for cleaning the necessary data before being put in a Hive or Impala table.

The data will be given as many relatively small files. The input data will be **xxxx.csv**-file for each sensor (e.g: **1763.csv**). Each of these files contains one row per measurement.

Example: The content of **1789.csv** looks similar to this:

ITE00100554,17890101,TMAX,-63,,,E,

ITE00100554,17890101,TMIN,-90,,,E,

GM000010962,17890101,PRCP,4,,,E,

EZE00100082,17890101,TMAX,-103,,,E,

EZE00100082,17890101,TMIN,-184,,,E,

ITE00100554,17890102,TMAX,-16,,,E,

ITE00100554,17890102,TMIN,-66,,,E,

GM000010962,17890102,PRCP,15,,,E,

EZE00100082,17890102,TMAX,-98,,,E,

EZE00100082,17890102,TMIN,-170,,,E,

Description of the columns:

* 1. o the sensor id
  2. o the date in format yyyymmdd
  3. o type of measurement
  4. o temperature in tens of degrees (e.g. -90 = -9.0 deg. C., -184 = -18.4 deg. C.)
  5. o other columns

**TASK 01**

**Import a File into HDFS**

1. Create a new directory in HDFS named /user/bcg/sensors

2. Put three files (xxxx.csv, xxxx.csv, xxxx.csv) from the /home/datasets/sensors directory on the local machine into /user/bcg/sensors directory in HDFS

In Terminal:

hdfs dfs -mkdir -p /user/bcg/sensors

hdfs dfs -copyFromLocal 17\* /user/bcg/sensors

**TASK 02**

**Cleanse Data using Pig**

Write a Pig script that satisfies all of the following criteria:

o Load all of the data in /user/bcg/sensors

o Remove all rows in the sensors data where the type of measurement column equals the string "TMIN".

o The output should only contain the SensorID, Date, TypeM’ and Temp

o Store the result as comma-separated records in a new directory in HDFS named /user/bcg/sensors\_clean

A = Load 'hdfs://localhost:8020/user/bcg/sensors/\*.csv' using PigStorage(',') as (SensorID:chararray, Date:chararray, TypeM:chararray, Temp:chararray);

B = Filter A By not (TypeM == 'TMIN');

STORE B INTO 'hdfs://localhost:8020/user/bcg/sensors\_clean';

**TASK 03**

**Define a Hive Table**

Define a Hive table named **sensors** that matches the data stored in your /user/bcg/sensors\_clean HDFS directory.

The Hive table should satisfy all of the following criteria:

o A Hive non-managed table with the location set to /user/bcg/sensors\_clean

o The schema matches the columns

1. SensorID string,

2. Date string,

3. TypeM string

4. Temp int

create database if not exists final;

use final;

drop table if exists sensors;

CREATE external table sensors

(SensorID string,

Date string,

TypeM double,

Temp int)

Row format delimited fields terminated by ',' stored as TextFile

TBLPROPERTIES ("skip.header.line.count"="1");load data inpath '/user/bcg/sensors\_clean/' into table sensors;

**TASK 04**

**Hive Partitioned Tables**

Define a new Hive-managed table named **sensors\_partitioned** that satisfies the following criteria:

o The table has the same schema as the sensors table

o The table is partitioned on the **Year** and **Month** columns

o Populate the table with data from the **2008\_bcg\_weather.csv** file into the appropriate partition of weather\_partitioned

In Terminal:

hdfs dfs -mkdir /user/bcg/sensors\_partitioned

hdfs dfs -put 2008\_bcg\_weather.csv /user/bcg/sensors\_partitioned

In HQL Script:

use final;

set hive.exec.dynamic.partition.mode = nonstrict;

drop table if exists sensors\_partitioned;

create table sensors\_partitioned (sensorid string, date string, typem string, temp int)

partitioned by (year string, month string)

row format delimited fields terminated by ','

stored as textfile

location '/user/bcg/sensors\_partitioned';

insert into sensors\_partitioned partition(year, month)

select sensorid, date, typem, temp, substr(date,1,4) as year, substr(date,5,2) as month

from sensors;

**TASK 05**

**Define and Populate an ORCFile Table**

Define a Hive table named **sensors\_orc** that satisfies all of the following criteria:

o A Hive-managed table

o The schema matches the columns in sensors table

o The table is stored as **ORCFile** format

o The table is populated with the records from the csv file 2019\_bcg\_weather.csv stored on HDFS

In Terminal:

hdfs dfs -mkdir /user/bcg/sensors\_orc

hdfs dfs -put 2019\_bcg\_weather.csv /user/bcg/sensors\_orc

In HQL Script:

use final;

set hive.enforce.bucketing = true;

set hive.enforce.sorting=true;

set hive.exec.dynamic.partition = true;

set hive.exec.dynamic.partition.mode = nonstrict;

drop table if exists sensors\_temp;

drop table if exists sensors\_orc;

CREATE external table sensors\_temp

(SensorID string,

Date string,

TypeM double,

Temp int)

Row format delimited fields terminated by ',' stored as TextFile

TBLPROPERTIES ("skip.header.line.count"="1");load data inpath '/user/bcg/sensors\_orc/' into table sensors\_temp;

CREATE external table sensors\_orc

(SensorID string,

Date string,

TypeM double,

Temp int)

stored as ORC;

INSERT OVERWRITE TABLE sensors\_orc SELECT \* FROM sensors\_temp;

drop table if exists sensors\_temp;

**TASK 06**

**Sqoop Export**

Assume the data is located into **/user/bcg/sensors/2019\_bcg\_weather.csv** on HDFS

1. Open a new MySQL session using the following command:

mysql -u root -p (password : cloudera)

2. Create a new database named **sensors**

create database sensors;

3. Switch to the sensors database

use sensors;

4. Create a new table named **weather** with the following schema:

+---------------+--------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+---------------+--------------+------+-----+---------+-------+

| stationID | varchar(100) | YES | | NULL | |

| year | varchar(15) | YES | | NULL | |

| typeM | varchar(10) | YES | | NULL | |

| temperature | int(11) | YES | | NULL | |

+---------------+--------------+------+-----+---------+-------+

5. Use Sqoop to export the **sensors** directory from HDFS to the **weather** table on MySQL.

i. For the credentials use the following:

username : root

password : cloudera

MySQL Shell:

mysql -u root -p

create database sensors;

use sensors;

create table weather (stationID varchar(100), year varchar(15), typeM varchar(10), temperature int(11));

grant all on sensors.\* to cloudera@localhost identified by 'cloudera';

Sqoop:

sqoop export -m -1 --connect jdbc:mysql://localhost/sensors --username root --password cloudera --table weather --export-dir /user/bcg/sensors/

**TASK 07**

**Flume Ingestion**

1. Create a Flume agent configuration file that satisfies the following criteria:

a) Having the Source defined as **spooldir** type which allows ingest data by placing files to be ingested into a “spooling” directory on disk.

b) The output data is written on HDFS into /**user/bcg/sensors\_in/** directory as a Hive partition. The partition key columns are Year, Month, Day.

2. Write the command line to run the Flume agent

In Terminal:

set hive.metastore.warehouse.dir=/user/bcg/sensors\_in/

Flume Agent Configuration File:

# Components

agt1.sources = source1

agt1.channels = mem\_channel

agt1.sinks = sink1

# Source Description

agt1.sources.source1.type = spooldir

agt1.sources.source1.spoolDir = /user/bcg/sensors\_spooldir

agt1.sources.source1.fileheader = false

agt1.sources.source1.fileType = DataStream

agt1.sources.source1.channels = mem\_channel

# Interceptor to add Timestamp

agt1.sources.source1.interceptor = ts

agt1.sources.source1.interceptor.ts.type = timestamp

# Channel Description

agt1.channels.mem\_channel.type = memory

agt1.channels.mem\_channel.capacity = 1000000

agt1.channels.mem\_channel.transactionCapacity = 100000

# Sink Description

agt1.sinks.sink1.type = hive

agt1.sinks.sink1.channel = mem\_channel

agt1.sinks.sink1.hive.metastore = thrift://127.0.0.1:9083

agt1.sinks.sink1.hive.database = final

agt1.sinks.sink1.hive.table = sensors

agt1.sinks.sink1.hive.partition = %Y,%m,%d

agt1.sinks.sink1.serializer = delimited

agt1.sinks.sink1.serializer.delimiter = ","

agt1.sinks.sink1.serializer.serdeSeparator = ','

agt1.sinks.sink1.serializer.fieldnames = SensorID, Date, TypeM, Temp

Run the Flume Agent:

flume-ng agent -n agt1 -f flume.conf -c conf