3.1 Files v/s Databases

Writeup (/writeup/3/1/) | Submissions (/submissions/3/1/) | ScoreBoard (/scoreboard/3/1/)

Phase	Open	Deadline
3.1 Files v/s Databases	Feb. 23, 2015 00:01	Mar. 01, 2015 23:59

Introduction

Learning Objectives

This project will encompass the following learning objectives:

1. Explore the advantages and disadvantages of utilizing flat files
2. Explore the advantages and disadvantages of utilizing databases
3. Explore the performance of vertical scaling in persistent cloud storage (magnetic vs SSD HDDs)

Introduction to Files and Databases

Files and databases are essentially resources for storing information that will be used by various computer programs. Files are mostly unstructured data, while databases organize data to make it easy to access efficiently for applications. To understand what unstructured or structured means, consider the following example which shows how the same line is stored and accessed from a file and a database.

The box below shows a single line in a file

```
Name: Carnegie, Course: Cloud Computing, Section: A, Year: 2015
```

The box below shows a single line (row) in a database table

```
|    Name    |      Course       |    Section     |    Year     |
|  Carnegie  | Cloud Computing |       A        |    2015     |
```

As you can see, in the case of a database, the data is represented in a more structured way which makes it easier for users of this data to access. In the case of a file, accessing the required data might require more effort from the end user.

The Scenario

Since you excelled at your job at MSB, you were hired for a part-time job by a music content delivery company, Carnegie Records (CR), as a music data analytics consultant. Before adopting cloud storage services, CR would like your help to evaluate the functionality and limitations in adopting flat files or relational databases to query and perform basic analytics on their data. You will commence your work with files and move on to databases.

As a first step in this process, CR has provided their dataset and a database as a saved instance on AWS. To get started, launch an instance on AWS. In the directory `/home/ubuntu/Project3_1/` , you will find two comma-separated files (csv), titled **million_songs_metadata.csv** and **million_songs_sales_data.csv**.

The million_songs_metadata.csv is all of the track metadata from the Million Song Dataset that CR uses and the **million_songs_sales_data.csv** has daily record sales count for CR for each song for a finite period of time. The two formats of the two files are as shown below.

Col #	Name	MySQL type
1	track_id	text(19)
2	title	text
3	song_id	text
4	release	text
5	artist_id	text
6	artist_mbid	text
7	artist_name	text
8	duration	double
9	artist_familiarity	double
10	artist_hotttnesss	double
11	year	int(11)

Table 1: MySQL Data Schema for million_songs_metadata.csv

Col #	Name	MySQL type
1	track_id	text(19)
2	sales_date	datetime
3	sales_count	int(11)

Table 2: MySQL Data Schema for million_songs_sales_data.csv

Resource Tagging

For this project, assign the tag with `Key: Project` and `Value: 3.1` for all resources

Operations on Flat Files

As a good consultant, you need to learn the tools necessary to carry out the job presented to you by CR. So, before you start working with the operations on flat files make sure you read about the grep (http://unixhelp.ed.ac.uk/CGI/man-cgi?grep) and awk (http://www.gnu.org/software/gawk/manual/gawk.html) tools in Unix. Now that you have quickly become an expert, start by launching an t1.micro instance of the following AMI.

Project AMI

For this part, the AMI ID is `ami-54590a3c`, and the instance type is `t1.micro`.

Task to Complete:

1.Use the following to find the records that contain `The Beatles`, while understanding the difference between the 2 commands below:

```
grep -P 'The Beatles' million_songs_metadata.csv
grep -i -P 'The Beatles' million_songs_metadata.csv
```

Make a note of the number of matches (Use `wc -l`) for the two commands in the step above for later use. Make sure to understand why there is a difference in the output of the two commands.

2.The result of running the grep command returns rows that include the search pattern in any matching column. What if you just wanted the ones that match the `artist_name` field? AWK is a data extraction and reporting tool. You can use awk to find the records with `artist_name` field containing `The Beatles`:

```
awk ' BEGIN {FS = ","} ; {if ($7 ~ /The Beatles/) { print; }}' million_s
ongs_metadata.csv
($7 denotes the 7th column and FS="," sets the field separator to a comm
a)
```

3.Now, what if we wanted to know something more complicated, like **"How many songs are from Michael Jackson, but only from the '80s ?"**

```
awk ' BEGIN {FS = ","} ; {if (tolower($7) ~ /michael jackson/ && $11 >=
1980 && $11 < 1990) { print; }}' million_songs_metadata.csv
(Column #11 is the year)
```

4.CR would like to know the average song duration in the dataset. They need you to write an awk program that computes the **average song length** (using the duration field) over the million songs. This program should be executed using a **single UNIX line**. Paste the command into the runner.sh, when asked.

5.CR would like you to implement the query to find "How many songs are from Michael Jackson? (Exact match)" using a shell script by looping through each record in the file (You can use `while`, `for` or `foreach` or any bash commands such as `grep` or `cut`). You will be asked to invoke the script in runner.sh.

6.CR would like to consolidate the data in the two source csv files mentioned earlier. To do that you need to either write a program or a set of commands (using awk or shell) to join the records from the **million_songs_metadata.csv** file and the **million_songs_sales_data.csv** file using the common column `track_id`.

The goal is to generate a new merged dataset `million_songs_metadata_and_sales.csv` with `track_id` as **column 1**, `sales_date` as **column 2**, `sales_count` as **column 3** and the other columns in million_songs_metadata.csv as **columns 4 to 13**. Save the output file and the program (or set of commands) for later use.

Hint: Search for a unix command that lets you join two files based on a common field.

You can see that as your questions become more complex, it is more difficult to write a single line of code to answer the question. For these reasons (among many others) we want to explore the benefits or limitations of **moving this data to a database**.

Working with MySQL

Carnegie Records (CR) would like its favourite consultant (you) to evaluate MySQL (an open source relational database) to load and process the data sets in preparation for querying and analytics so that it can decide whether to adopt flat files or databases for its needs.

Project AMI

For this part, the AMI ID is `ami-54590a3c`, and the instance type is `t1.micro`.

MySQL should be already installed and running on the AMI supplied for this project. To start a MySQL CLI client and connect to the running MySQL server on your instance, use the following command:

```
mysql -u root -pdb15319root song_db
```

In the command above, the username is **root** and the password is **db15319root**. The database that you will be using is **song_db**, which has been already created for you within the AMI.

You should now take a closer look at databases and what a database schema is. A database holds the records, fields and cells, of data while a database schema describes how these fields and cells are structured and organized and what types of relationships are mapped between these entities. A database can have multiple tables that are related to each other. A table is referred to as a relation in the sense that it is a collection of objects of the same type (rows). The schema defines the structure of the tables and the relations between them.

Before loading CR's records into the database, we need to describe the database and its fields in a schema, which encapsulates the structure and relationship of the data. CR has provided two csv files, million_songs_metadata.csv and million_songs_sales_data.csv. CR also provided their schemas in Table 1 and Table 2 described before.

CR has asked you to create two tables within the database according to the schemas. They have provided a `~/Project3_1/create_tables.sql` file for you to get started.

Note

- MySQL has some keywords that cannot be used in queries. When a table or column name contains a keyword use the `backtick (`)` as an escape sequence. Example - `` `keyword` ``
- The MySQL package pre-installed in the AMI provided is version 5.5. Make sure you search through the documentation for the correct version when looking for command references.

After creating the tables (songs & sales), you can use the following command to verify the schema of that table:

```
DESCRIBE songs;
DESCRIBE sales;
```

After executing these commands, you should compare and verify that they match with the information provided in tables 1 and 2 above.

You now need to find the appropriate command to load the database from the supplied files, `million_songs_metadata.csv` and `million_songs_sales.csv` . You can use either a **SQL command from within the MySQL CLI client**, or use the **mysqlimport utility** to load the records from the CSV file into the database. Please note down the command you have used.

To verify whether the data has been successfully loaded into the database, you can list the first 10 records of the table using the SELECT command:

```
SELECT * FROM songs LIMIT 10;
```

The equivalent SQL command to find the records that contain `The Beatles` (equivalent of the Beatles awk command from the previous part) is:

```
SELECT * FROM songs WHERE artist_name LIKE '%The Beatles%';
```

Similarly the Michael Jackson query would look like:

```
SELECT * FROM songs WHERE artist_name LIKE '%michael jackson%'AND year >= 1980
 AND year < 1990;
```

While evaluating flat files you have used **awk** to compute the average song length (using the duration field) over the million songs. The equivalent SQL command is:

```
SELECT AVG(duration) FROM songs;
```

Now you have successfully built the tables and made simple queries. Since the dataset is quite large, CR wants you to ensure that the database responds rapidly to queries. As an informed consultant, you will consider creating an indexed database to improve performance. A database index helps speed up the retrieval of data from tables. When you query data from a table, MySQL checks if the indexes exist. Then MySQL uses the indexes to select exact physical corresponding rows of the table instead of scanning the whole table. You conclude that you should create an index on the columns of the table from which you often query the data. Notice that all primary key columns are in the primary index of the table automatically. In the following steps, you are going to compare the MySQL query response times before creating the index with the time recorded after creating the index. Take note of the following four MySQL query response times before indexing:

```
SELECT * FROM songs WHERE duration=(SELECT MAX(duration) FROM songs);
SELECT * FROM songs ORDER BY duration DESC LIMIT 5;
SELECT * FROM songs WHERE duration=(SELECT MIN(duration) FROM songs);
SELECT * FROM songs ORDER BY duration ASC LIMIT 5;
```

Now you want to create an index for the table songs. Create an index for one column to improve the speed of the queries above. Which column do you think you should create the index for? Create an index on the column you think is the most reasonable. Please note down the command you have used when creating the index and paste it into the runner.sh file when asked. Creating an index might take a while since MySQL has to build and maintain the index table. The payoff is in speeding up all subsequent queries that could take advantage of the index. After that has finished, exit MySQL and restart it using the following command:

```
sudo service mysql restart
```

Then relaunch the mysql client, and re-run the four SQL queries above, noting the new times ("after index") for the quiz.

Note

Please update the variable `"INDEX_NAME"` in runner.sh with your index name before running the script.

Sometimes SQL queries are more complicated than their grep/awk counterparts. Write SQL queries that will return the same results as the following commands you used while evaluating flat files. Note each query for the quiz.

```
grep -P -c 'The Beatles' million_songs_metadata.csv
grep -i -P -c 'The Beatles' million_songs_metadata.csv
```

Hint: The SQL LIKE operator is case insensitive by default.

Note

You are going to perform some SQL queries in the checkpoint quiz related to `JOIN` and `GROUP BY` using the same two tables. If you do not want to lose data, do not terminate your instance until you finish the runner.sh, or take a snapshot of your current instance to save everything.

After you have experienced the basics of MySQL, it is time to move forward to **Aggregate Functions**.The aggregate functions allow you to perform calculations on a set of records and return a single value. The most common aggregate functions include `SUM, AVG, MAX, MIN and COUNT`. Aggregate functions are often used with the MySQL GROUP BY keyword to perform calculations on each subgroup and return a single value for each subgroup. The MySQL GROUP BY keyword is used with the SELECT statement to group rows into subgroups by one or more columns or expressions. It is extremely useful when several records belong to a category and other records in the same table belong to another category and you want to compare between different categories rather than a single record. The following statement illustrates the MySQL GROUP BY keyword syntax:

```
SELECT c1, c2, … cn, aggregate_function(expression)
FROM table
WHERE where_conditions
GROUP BY c1, c2, … cn;
```

For example, the query about the total sales for each of the recent 10 days looks like:

```
SELECT sales_date, SUM(sales_count) AS total_sales
FROM sales
GROUP BY sales_date
ORDER BY sales_date DESC
LIMIT 10;
```

The MySQL JOIN keyword is used to query data from two or more related tables. In MySQL, `JOIN,CROSS JOIN and INNER JOIN` are syntactic equivalents (they can replace each other). The following illustrates a sample JOIN syntax:

```
#select statement
    SELECT c1,c2,....cn
    FROM join_table;
#join_table
    table1 [INNER|CROSS] JOIN table2 [join_condition]
#join_condition:
    ON conditional_expr
    USING (column_list)
```

INNER JOIN builds the Cartesian product between specified tables, that is, each and every row in the first table is joined to each and every row in the second table based on the join condition. Here all the track_ids in table songs have corresponding records in table sales. We are going to only cover INNER JOIN.

What about OUTER JOIN?

OUTER JOIN identifies rows without a match in the joined table. Outer joins combine two or more tables in a way that some columns may have NULL values. MySQL separates OUTER JOINS into LEFT or RIGHT JOINS depending on which table provides the unmatched data. In a LEFT JOIN, the unmatched records from the table on the left side of the JOIN clause are returned. In a RIGHT JOIN, the unmatched records from the table on the right side of the JOIN clause are returned. When no match was found, MySQL sets the value of columns from the joined table to NULL. This is important for example when you want to select all records from a reference table that have no related data in another.

For example, the query that shows the title of the song that has the most sales count looks like:

```
SELECT songs.title, SUM(sales_count) AS total_sale FROM songs INNER JOIN sales
 ON songs.track_id = sales.track_id GROUP BY sales.track_id ORDER BY total_sal
e DESC LIMIT 10;
```

Now that you have explored both files and databases, you can realize that both of these storage systems have advantages and disadvantages. Files can be flexible and portable, can hold structured and unstructured data, and is easy to implement and modify. Databases on the other hand can represent relationships among data, and has easy rollback of transactions in case of failure. Moreover, there are more advantages and disadvantages about files and databases waiting for you to explore. For files, security can be achieved only through file permissions while databases supports fine grain level security. The access mode for files is sequential, thus it is slower when the files are larger. But for databases, it is more concurrent. Further, reads and updates can be rule-based in databases using constraints which ensures data consistency. When updating databases, a specific record can be updated, on the other hand, file contents have to be scanned when updating data. Furthermore, you can easily rollback transactions in case of a failure in a database while you cannot do that with files. Databases have to maintain transaction consistency which is expensive.

Vertical Scaling

In the following sections, you will get acquainted with common disk operations in Linux and use those commands to perform vertical scaling of storage devices by attaching different storage devices offered in AWS to an instance and measuring the performance using some common benchmarking tools.

Project AMI

For this part, the AMI ID is `ami-54590a3c` , and the instance types to be used are `t1.micro` and `m3.large` .

Common Disk Operations in Linux

On this page there are several mini-howto's for accomplishing certain disk-related tasks in Linux, similar to the ones you have seen in the Project Primer.

Since most of these commands require root access, you might find it easier to run the following once: `sudo su` or alternatively you can prepend sudo to each command.

Working with EBS Volumes

The following video will demonstrate the use of EBS volumes with EC2 instances:

Video 1: EBS volumes with EC2 instances

Show available drives

GNU **parted** is a program for creating, destroying, resizing, checking and copying partitions, and the file systems on them. The partition editor, **parted**, is used to manipulate the partition table on block devices. It has replaced **fdisk** and supports newer features like GUID Partition Tables (GPT), which are required for volumes > 2TB. You can also run **parted** with no parameters to get interactive mode. You can read more from here.
(http://www.gnu.org/software/parted/manual/html_chapter/parted_1.html)

```
parted -l
```

/dev/xvda1 – this is the OS partition

/dev/xvdb – this is the first Ephemeral (instance store) drive

/dev/xvdc – this is the second Ephemeral (instance store) drive

Create and format a partition

```
umount /dev/xvdX #where “X” – is a,b,c..etc (You should use your device’s name
)
parted /dev/xvdX mklabel gpt
parted /dev/xvdX mkpart db ext4 0% 10G
mkfs.ext4 /dev/xvdX1
```

If you have a small volume (such as EBS), it’s easier to just format the whole device directly, without creating a partition table (note the missing ‘1’):

```
mkfs.ext4 /dev/xvdX
```

Mount a volume

```
mkdir /storage/mountpoint
mount /dev/yourdevice /storage/mountpoint
```

mountpoint is your choice, but you have to use a different name for each filesystem that is mounted at the same time. (/storage/ is arbitrary as well – most Linux mount extra drives under /mnt, but this automatically mounts the first ephemeral drive at boot on EC2 images). yourdevice is the device or partition that you have previously formatted, such as xvdX1, md0p1, etc.

Note

Running mount without parameters shows all mountpoints. Use this to verify that your mount was successful.

Running sysbench for FileIO

In this project, you will be comparing the SSD and magnetic storage types using **sysbench**, a suite containing multiple benchmarks. For this task, you will be using a modified version of sysbench with identical functionality as the original one except that you can declare the storage type as [SSD|Magnetic] when “sysbench run” command is called. To test disk performance, we can use the FileIO benchmark. The benchmark can be run by doing the following:

Sysbench

1. The first step is to create a test file dataset. If you are using the course AMI for the project, you can run the following command:

   ```
   sudo /home/ubuntu/Project3_1/sysbench --test=fileio --file-total-si
   ze=100G prepare
   ```

   The above command will generate a test file set of a total of 100 GBs. When you run this command make sure you run it after changing the directory into the folder where your storage device is mounted.

2. Once the test files are deployed, you can run the benchmark using the following command. Use SSD or Magnetic as the first argument to specify your disk type:

```
sudo /home/ubuntu/Project3_1/sysbench [SSD|Magnetic] --test=fileio
--file-total-size=100G --file-test-mode=rndrw --max-time=300 --max-
requests=0 run
```

   Hint: This command should be run under the directory with test files (the same directory you execute sysbench prepare command).You can read more about sysbench here (http://wiki.gentoo.org/wiki/Sysbench).

FileIO Performance Benchmarks

In this part of the project, we will be experimenting with "Vertical Scaling" for storage and instance types, by attaching Magnetic (i.e. spinning disks) and Solid State Drive (SSD) storage devices offered in AWS to two types of instances and measuring their performance for FileIO.

As a result of this project exercise, we expect you to gain insight on the benefits of employing faster storage systems such as SSD and using larger instances.

Project AMI

AMI is `ami-54590a3c`. We want you to test the following scenarios:

Scenario	Instance Type	Storage Type
1	t1.micro	EBS Magnetic Storage
2	t1.micro	EBS General Purpose SSD
3	m3.large	EBS Magnetic Storage
4	m3.large	EBS General Purpose SSD

Experiment 1

To run the benchmark for this experiment, perform the following steps:

1. Launch a t1.micro instance with the AMI given above.
2. Attach a 200 GB EBS volume of the desired type (Magnetic or SSD EBS) using the EC2 console.
3. SSH to the instance.
4. Perform the required steps to format and mount the EBS volume into the filesystem on

that instance.
5. Use the **sysbench fileio test** command with the `prepare` option to generate 100 GB of data into the attached EBS volume.
6. Run **sysbench fileio test** on the data three times (without allowing for a long delay between runs)
7. Enter your answers in runner.sh.
8. Repeat steps 2 to 6 for the next storage type.

Do not delete the EBS volumes at this point.

Experiment 2

Please read the following instructions carefully. They are not the same as in Experiment 1.

You can reuse the test files generated by sysbench prepare command from the previous experiment. So please don't format the disks from the previous experiment, or you will lose the data.

1. Launch a m3.large instance with the AMI given above.
2. Detach the two 200 GB EBS volumes you used in previous steps.
3. Attach one of the 200 GB EBS volumes of the desired type (Magnetic or SSD EBS) using EC2 console.
4. SSH to the instance and mount the EBS volume into the filesystem on that instance.
5. Run **sysbench fileio test** on the data three times (without allowing for a long delay between runs).
6. Record your results for the quiz.
7. Repeat step 3 to 6 for another storage type.

Note

If you want to redo the experiments, please reboot the instance from AWS console. After rebooting, you need to mount the disk again. But don't format the disk or generate test files using the sysbench prepare command.

Grading

To complete the project, you are expected to answer some questions provided in the same AMI. The quiz questions are present in the file `/home/ubuntu/Project3_1/runner.sh`. You can verify and submit your results using the given auto-grader in the AMI. To use the autograder, do the following:

1. Go to the auto-grader folder located at `/home/ubuntu/Project3_1`
2. The auto-grader consists of three files, `runner.sh`, `submitter` and `references`. You have permissions to edit `runner.sh` and `references` files.
3. Edit the script `runner.sh` to include the commands/code used to answer the questions. You need to complete the questions using only bash commands and MySQL queries. Using any language other than bash will be given a grade of 0. Do not move any of the provided files. If you are using any external scripts, ensure that you are calling the correct scripts from `runner.sh`. Please ensure that you are placing all your code in the same folder and also assume that the dataset is present in the current folder.
4. Edit the text file `references` to include all the links that you referred to for completing this project. Also include the Andrew IDs of all the other students who you might have discussed general ideas with when working on this project in the same file.
5. You can run the autograder by typing `./runner.sh` from the autograder folder. Running this script should print out the answers to all the questions. Please ensure that the answers are printing correctly before using submitter.
6. Once you have completed all the questions, you can submit the answers to the evaluation system using the auto-grader executable `submitter`. Run the executable using the command `./submitter` from the autograder folder. After running this command, you should be able to see your scores on the website in a few minutes. There is no limit on the number of submissions allowed before the project deadline. However, each submission must be separated by at least 60 seconds.
7. NOTE: There might be some questions in the `runner.sh` which will be manually graded at the end of this project.

Project Grading Penalties

Besides the penalties mentioned in recitation and/or on Piazza, penalties accrue for the following:

Violation	Penalty of the project grade
Spending more than $15 for this project checkpoint	-10%
Spending more than $30 for this project checkpoint	-100%
Failing to tag all your resources for this project	-10%
Using any instance other than t1.micro or m3.large	-10%
Attempting to hack/tamper the auto-grader	-100%