Apache Spark, Real Time Data Pipelines and Analysis

**Table of Contents**

[Section 1: Task Description 2](#_Toc475459847)

[1.1 Task Deliverable Details 3](#_Toc475459848)

[Section 2: Spark Design 4](#_Toc475459849)

[Section 3: Application Queries and Outputs 5](#_Toc475459850)

[3.1 Word Count 5](#_Toc475459851)

[3.2 Analysis of Baby Names Dataset 7](#_Toc475459852)

[3.2.1 Total number of birth registered in a year. 7](#_Toc475459853)

[3.2.2 Total number of births registered for a year by gender 8](#_Toc475459854)

[3.2.3 Top 5 most popular names registered for a year 9](#_Toc475459855)

[3.2.4 Total number of birth registrations for a name 9](#_Toc475459856)

[3.3 Analysis of NYPD Motor Vehicle Collision Dataset 10](#_Toc475459857)

[3.3.1 Total injuries and fatalities 10](#_Toc475459858)

[3.3.2 Total incident in a year 12](#_Toc475459859)

[3.3.3 Total injuries grouped by year and quarter 13](#_Toc475459860)

[3.3.4 Total incidents grouped by borough, year and month 15](#_Toc475459861)

[Section 4: Summary 17](#_Toc475459862)

[4.1 Comments on Apache Spark 17](#_Toc475459863)

[4.2 Observations and Recommendations 17](#_Toc475459864)

[References 18](#_Toc475459865)

# Section 1: Task Description

We will learn the concepts of Big Data Systems running on Clouds. We will use Apache Spark to build Real Time Data Pipelines which can be used over data-warehouse and distributed databases. Apache Spark is a cluster computing framework which allows lighting fast computing through in-memory computing, along with features of implicit data parallelism and fault tolerance [1]. It has Application Programming Interfaces in R, Python, Java and Scala through which it offers distributed task scheduling, dispatching and basic I/O functions [1]. The following diagram illustrates an overview of Apache Spark.

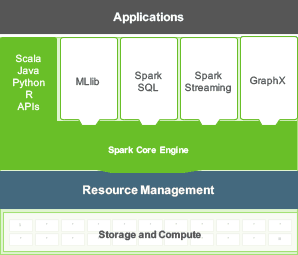


Fig 1.1: Apache Spark Architecture overview. [2]

Apache Spark can run Spark locally with one or more worker thread. It can also connect to a Spark Standalone or a Meso cluster using IP and Port for cluster computing. Apache Spark uses RDD (Resilient Distributed Dataset) to have immutable, partitioned and distributed in-memory storage of data from different sources to perform data transformations and actions on them.

We will use local standalone configuration to perform following data-analysis:

1. **Word Count**: Count the number of distinct words in a text document.
2. **SQL like operations on Big Datasets**: Loading data into Spark and performing SQL like operations on it using Spark SQL and Dataframes.
3. **Advanced SQL operations (aggregations, Roll-up, drill down etc.)**: Loading data into Spark, performing required pre-processing and further deriving insights using aggregations, Roll-up and drill down like operations on the dataset.

## 1.1 Resource Details

The following table lists all other resources submitted along with this report:

|  |  |  |
| --- | --- | --- |
| Serial No | Resource | Path |
| 1. | Word Count Application and output for analysis on Divine Comedy paragraph [3]. | word\_count |
| 2. | Application and sample output for analysis on National Names dataset [4]. | baby\_names\_analysis |
| 3. | Application and sample output for analysis on NYPD Motor Vehicle collision dataset [5]. | nypd\_mv\_collision\_analysis |
| 4. | Readme File | README.md |

Table 1.1.1: Resource Details

# Section 2: Spark Design

For the current task, a standalone deployment of Spark on local machine was used. We used a Spark deployment on a Windows 7 operating system and local file system for storing data from where the data was loaded to generate RDD. The diagram below shows the deployment overview:

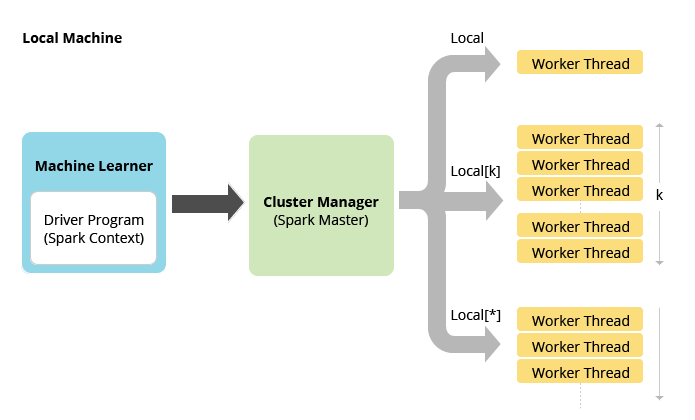


Fig 2.1: Standalone deployment mode of Apache Spark [6]

For the above deployment scenario, Apache Spark runs in local mode and can have one or more than one worker threads on the same machine. The number of worker threads should be ideally set to the number of cores on the machine and is configured as follows while creating the Spark context.

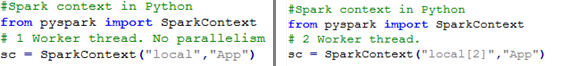


Fig 2.2: Configuring Spark Context on standalone deployment

We used Spark version 2.1.0. The following table shows the addition configurations done to setup Spark on Windows operating system.

|  |  |  |
| --- | --- | --- |
| Serial No. | Path Variable | Value |
| 1 | JAVA\_HOME | C:\Program Files\Java\jdk1.7.0\_79 |
| 2 | SCALA\_HOME | C:\Program Files\scala |
| 3 | SBT\_HOME | C:\Program Files\sbt |
| 4 | SPARK\_HOME | C:\spark-2.1.0-bin-hadoop2.7\ |
| 5 | HADOOP\_HOME | E:\winutils |
| 6 | PATH | %SCALA\_HOME%\bin; %SBT\_HOME%\bin;  %SPARK\_HOME %\bin;%JAVA\_HOME%\bin;%HADOOP\_HOME%\bin |

Table 2.1: Configurations to set-up Spark on Windows [7]

We used Jupyter Notebook [8] provided by Anaconda [9] installation package as IDE to write program.

# Section 3: Application Queries and Outputs

In the following sub-sections, we will see the applications queries and outputs for different datasets.

## 3.1 Word Count

**Requirement:** Write an application which can count distinct words and number of occurrences of each word in the dataset.

The following approach was adopted:

1. We used the content of ‘*WordCountData.txt*’ as the input dataset.
2. Data Preprocessing:
   1. Removed special characters like comma, empty spaces etc.
   2. Removed stop words using Natural Language Tool Kit.
3. Further we used Apache Spark’s *flatMap* and *reduceByKey* methods to build a dictionary of unique words.
4. We also used Pandas package to generate a bar plot for 20 most frequent important words.

Below is the snapshot of the application for word count:



Fig 3.1.1: Application to count unique words

Below is the snapshot of the partial-output of the application along with the processing time:

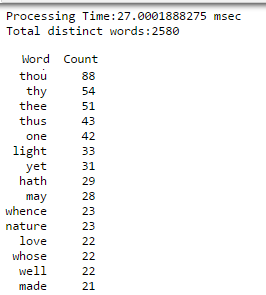


Fig 3.1.2: Output of application for word count

Below is the snapshot of the bar-chart for the top 20 words generated using Pandas.

Query Processing Time is **27 Milliseconds.**

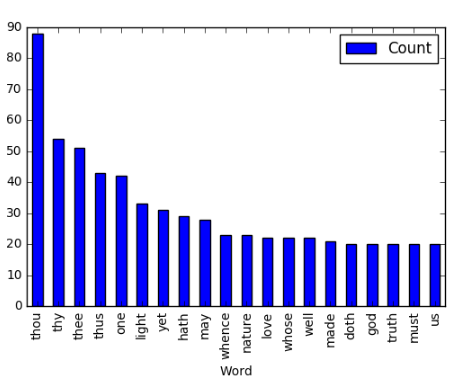


Fig 3.1.3: Bar Chart for top 20 words generated using Pandas.

## 3.2 Analysis of Baby Names Dataset

**Requirement:** Import the data from baby names dataset and write an application which can be used to perform queries.

The following approach was adopted:

1. NationalNames.csv has been used as the input dataset.
2. Data has been imported into dataframe using *sqlContext.read.load* function.
3. *sqlContext.sql* was used to query data from the dataframe.

### 3.2.1 Total number of birth registered in a year.

Below is the snapshot of the application:

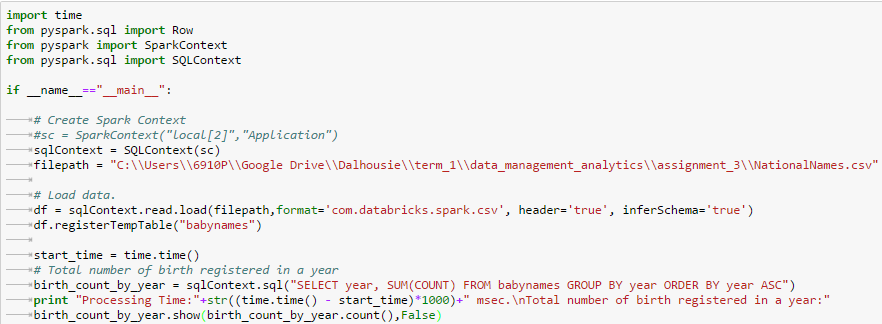


Fig 3.2.1.1: Application to count total birth registered in a year.

Below is the snapshot of the partial output. Complete output is shared in *src/baby\_names\_analysis/2.1.txt* file.

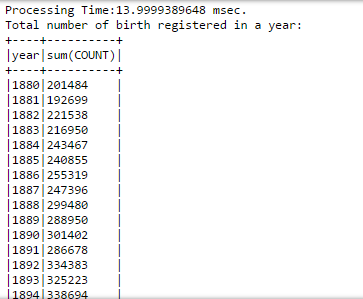


Fig 3.2.1.2: Partial output having count of total birth registered in a year.

Query processing time is **14 Milliseconds.**

### 3.2.2 Total number of births registered for a year by gender

Below is the snapshot of the application:



Fig 3.2.2.1 Sample application to count births registered for a year by gender

Below is the snapshot of the partial output. Complete output is shared in *src/baby\_names\_analysis/2.2.txt* file:

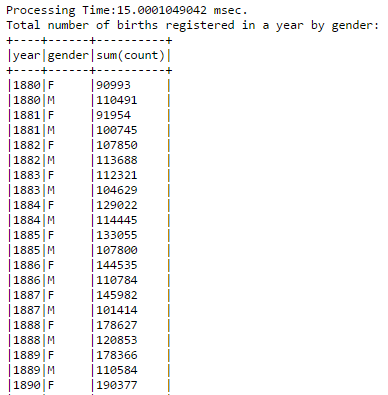


Fig 3.2.2.2: Output for the above query

Query processing time is **15 Milliseconds**

### 3.2.3 Top 5 most popular names registered for a year

Below is the snapshot of the application and the result:

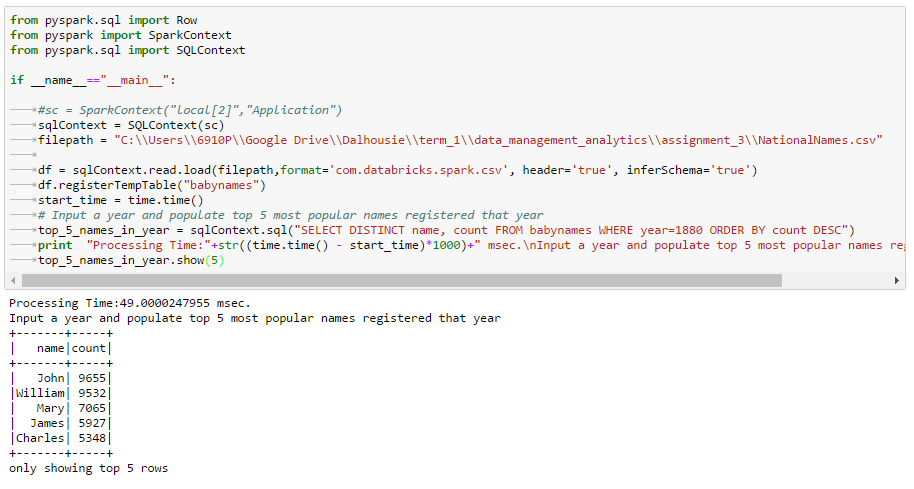


Fig 3.2.3.1: Sample application and output for 5 most popular names of year 1880

Query processing time is **49 Milliseconds.**

### 3.2.4 Total number of birth registrations for a name

Below is the snapshot of the application:



Fig 3.2.4.1: Sample Application to get total birth registered for a name

Below is the snapshot of the output:

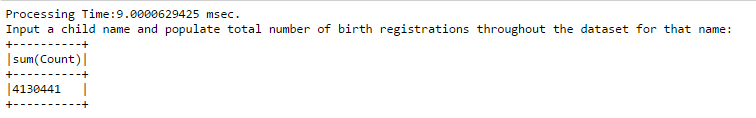


Fig: 3.2.4.2: Output for total birth registered for the name ‘Mary’

Query processing time is **9 Milliseconds**.

## 3.3 Analysis of NYPD Motor Vehicle Collision Dataset

**Requirement:** Import the data from NYPD dataset and write an application which can be used to perform the queries.

The following approach was adopted:

1. Data preprocessing
   1. Updated the column header names where empty spaces in names were replaced by “\_”.
   2. Corrected a corrupted row where the data for one column was split across two lines.
2. Data frame was updated with additional columns to hold year, month and quarter (as required by 2nd query) and month (as required by 3rd and 4th query) from the date column of the provided dataset.
   1. A function to get year from date was implemented.
   2. A function to get month from date was implemented.
   3. A function to get quarter from date was implemented.
3. *sqlContext.read.load* was used to load data into the dataframe.
4. Application also provides the feature to load the dataset from local machine or from web hosted resource.
5. *sqlContext.sql* was used to query data from the dataframe.

### 3.3.1 Total injuries and fatalities

**Requirement:**  Capture total injuries and fatalities associated with each motor collision record, identified by a unique incident key.

Below is the snapshot of the application:



Fig 3.3.1.1: Application to count total injuries and fatalities

Below is the snapshot of the partial output. Detailed output is shared in ./*src/ nypd\_mv\_collision\_analysis/3.1.txt* .

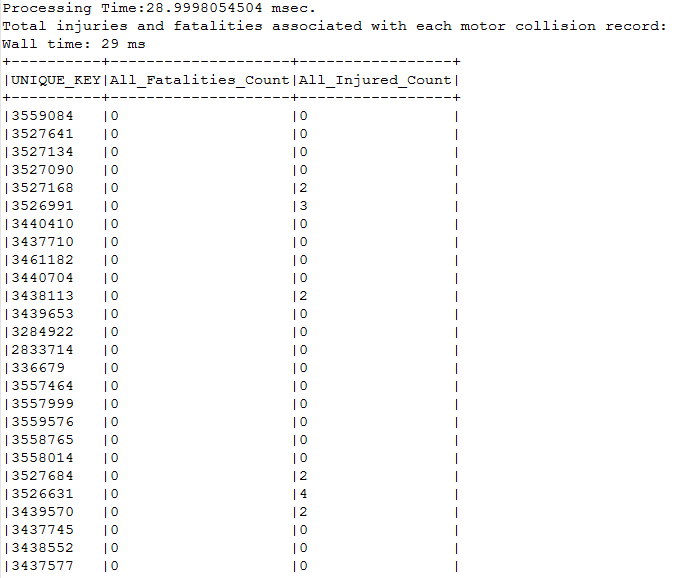


Fig: 3.3.1.2: Output for count of total injuries and fatalities

Query processing time is **28.99 Milliseconds.**

### 3.3.2 Total incident in a year

**Requirement:** Capture total incident counts in a year (grouped by year)

Below is the snapshot of the application:



Fig: 3.3.2.1: Sample application to count incidents in a year

Below is the snapshot of the output:

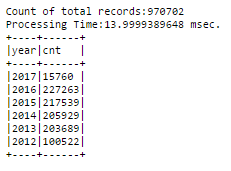


Fig: 3.3.2.2: Output for count of incidents in a year

Query processing time is **13.99 Milliseconds**.

### 3.3.3 Total injuries grouped by year and quarter

**Requirement:** Capture total injuries (can be sum of injuries and fatalities) grouped by year and quarter

Below is the snapshot of the application:



Fig 3.3.3.1: Sample application to count injuries grouped by year and quarter

Below is the snapshot of the output:

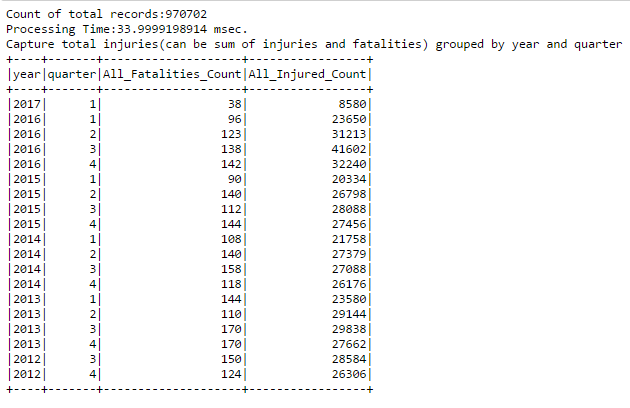


Fig: 3.3.3.2: Output for count of injuries grouped by year and quarter

Query processing time is **33.99 Milliseconds**.

### 3.3.4 Total incidents grouped by borough, year and month

**Requirement:** Capture total injuries (sum of injuries and fatalities) and incident count grouped by Borough, year and month

Below is the snapshot of the application:



Fig: 3.3.4.1: Sample application to get total incidents grouped by borough, year and month

Below is the snapshot of the output:

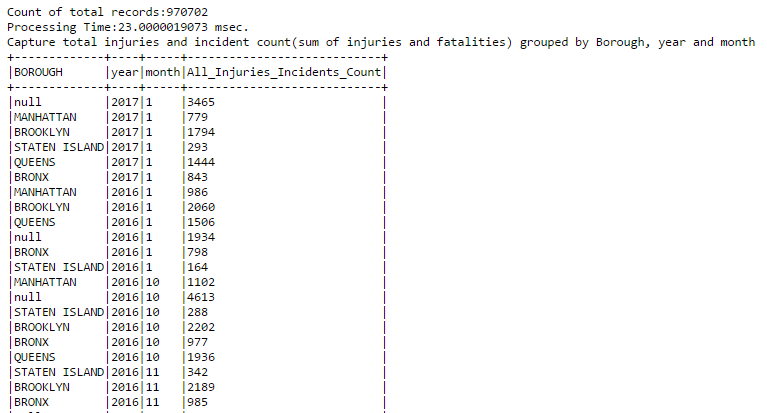


Fig: 3.3.4.2: Sample output for total incidents grouped by borough, year and month

Query processing time is **23 Milliseconds**.

# Section 4: Summary

## 4.1 Comments on Apache Spark

We used Apache Spark in standalone deployment mode on a Windows 7 operating system. It has a nice integration with development and analytics packages like Anaconda, Jupyter and Pandas which simplifies the data analysis by providing interfaces for development, debugging and visualization to gain quick and better insights in the data. Data of varied formats (CSV, JSON etc) can be easily imported and analyzed. Also, with its API in a variety of languages like R, Java, Scala and Python, it is easy to adopt.

## 4.2 Observations and Recommendations

We used Apache Spark to perform data analysis. Using Apache Spark, we were able to perform operations like couniting unique words, perform basic SQL like operations and advanced operations like Roll-up and drill down in real time. The following table summarizes the performance observed.

|  |  |  |  |
| --- | --- | --- | --- |
| Serial No. | Data Set | Dataset Size | Performance Measure |
| 1. | WordCountData.txt | 60KB,  ~1500 Lines | Counted all words in **27 milli-seconds** |
| 2. | NationalNames.csv | 42MB,  ~2 Million Records | Average Query Time: **11 milli-seconds**  (For 4 queries discussed in Section 3) |
| 3. | NYPD\_Motor\_Vehicle\_Collisions.csv | 176 MB,  ~1 Million Records | Average Query Time: **22 milli-seconds**  (For 4 queries discussed in Section 3) |

Table 4.1: Evaluation of query performance on Apache Spark

On the basis of above observations, we can conclude that Apache Spark can be used for real time data analysis. Through its in-memory computation the data analysis is quick and efficient. We could also use it to manipulate data like adding additional columns required for aggregations (like roll up by quarter or month) which are very important to gain in-depth insights.

# References

|  |  |
| --- | --- |
| [1] | "Apache Spark," 20 February 2017. [Online]. Available: https://en.wikipedia.org/wiki/Apache\_Spark. |
| [2] | "What is Apache Spark," Hortonworks, 2017. [Online]. Available: http://hortonworks.com/apache/spark/. [Accessed 20 February 2017]. |
| [3] | "Divine Comedy: Paradiso E-Text | Canto 1," Gradesaver.com, [Online]. Available: http://www.gradesaver.com/divine-comedy-paradiso/e-text/canto-1. [Accessed 21 February 2017]. |
| [4] | "US Baby Names | Kaggle," Kaggle.com, [Online]. Available: https://www.kaggle.com/kaggle/us-baby-names. [Accessed 21 February 2017]. |
| [5] | "NYPD Motor Vehicle Collisions | NYC OpenData," Data.cityofnewyork.us, [Online]. Available: https://data.cityofnewyork.us/Public-Safety/NYPD-Motor-Vehicle-Collisions/h9gi-nx95. [Accessed 21 February 2017]. |
| [6] | "Deployment Patterns - Machine Learner 1.1.1 - WSO2 Documentation," Docs.wso2.com, [Online]. Available: https://docs.wso2.com/display/ML111/Deployment+Patterns. [Accessed 20 February 2017]. |
| [7] | N. Tayal, "How to run Apache Spark on Windows7 in standalone mode," Nishutayaltech.blogspot.ca, [Online]. Available: http://nishutayaltech.blogspot.ca/2015/04/how-to-run-apache-spark-on-windows7-in.html. [Accessed 20 February 2017]. |
| [8] | "Installing Jupyter Notebook — Jupyter Documentation 4.1.1 alpha documentation," Jupyter.readthedocs.io, [Online]. Available: http://jupyter.readthedocs.io/en/latest/install.html. [Accessed 20 February 2017]. |
| [9] | "Anaconda Overview," Continuum, [Online]. Available: https://www.continuum.io/anaconda-overview. [Accessed 20 February 2017]. |