# Efficient Product Inventory Maintenance for Black Friday Sale via Spark Big Data System

**Project Report for Course IS 789 (Big Data Fundamentals and Techniques), Fall 2019**

Chhaya Kulkarni, Kavitha Loganathan Sundara Rajan and Xin Wang

Department of Information Systems, University of Maryland, Baltimore County

**Abstract**

Black Friday fever attracts customers from far and wide. Excitement also leads to frantic chaos for e-commerce websites. To help ease the burden on the e-commerce companies managing the traffic as well as inventory is of utmost importance. An environment that emulates the Black Friday sale can be created well using Spark Streaming component. Spark SQL and Dataframes can be used to carry out operations and store data. We have attempted to see how we can effectively handle the inventory using Spark components.

## 1. Introduction

We are planning to ingest data into the cluster using Spark Data streaming component. Dataset that we are going to be uploading into the Spark cluster will hold Product and customer information. The attributes of these data are for example: the description, prices, unique identifier, product quantity, stock, etc. of the products and the address, phone number, zip code, etc. of the customers. An order data set will also be used in our project.

These datasets are defined as the raw data set. Both the datasets will be transferred to data frame in Spark. Not only storing these raw data in Spark HDFS, we will also be modifying the datasets that we have downloaded. Datasets will be updated by the application that we are building to help maintain the inventory.

We are planning to maintain the inventory of products as and when products are sold during the Black Friday sale. This will help the company monitor which products can still be displayed as available on sale. This will help both the customers and e-commerce website understand what the bestselling products are. Companies can accordingly update their stock and customers can also plan on which product should they add to their shopping cart faster.

## 2. Motivation

During Black Friday sales, there will be a huge number of users trying to buy items in a limited time deal online. When the time started, thousands of customers log into the website for the limited number of products. Currently, it is important for organizations to be able to handle that huge payload of users. If the site crashes down, then it will cause damage to the company's reputation. It can lead to loss of valuable customer base too.

In order to overcome this, it is imperative for the company to design an efficient pattern that utilizes resources in a balanced and fair manner. Customer satisfaction is also dependent on this factor. Customer wait period should be restricted to a minimum time as is possible. Balancing the usage of resources can also result in saving millions of dollars for the company and boost its profit rates. These three major reasons have made us realize how important it is to design an efficient system data flow pattern.

## 3. Problem Statement

Maintaining inventory is a crucial aspect in running any business. Stock availability ensures customer satisfaction. We are trying to address the problem of stock update when orders are placed by customers. Every time an order is placed; we check the stock count of the products mentioned in the order. If there is enough availability, then we update the stock.

We validate User ID to ensure authentication of the user. We validate Product ID to verify whether the mentioned product is enlisted with the e-commerce company. We check the stock to see if enough quantities of products are available and then decide if the order can be fulfilled or not.

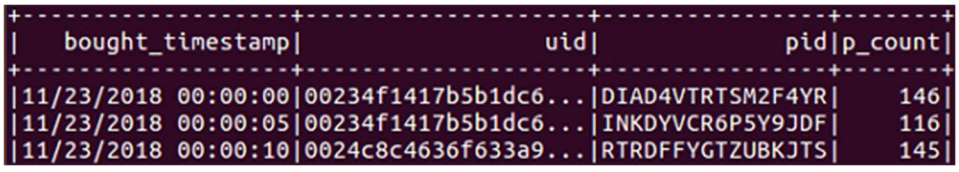
## 4. Dataset

For our project, we used three datasets user, product and orders. Orders dataset was streamed it contained combination of user and products.

Table 1: Dataset Summary

| | |
|---|---|
| Datasets Used: | Product, User, Order |
| Description: | Product dataset was created so that customers could buy items during sale. User dataset was used to simulate customers. |
| Number of records: | User dataset contains 5K records<br>Product dataset contains 5K records<br>Order dataset contains 13K records and is used for streaming |
| Link: | https://www.kaggle.com/PromptCloudHQ/flipkart-products<br><br>https://www.kaggle.com/rtatman/every-pub-in-england |

Sample of the datasets used in our application looks as shown below:



| bought_timestamp | uid | pid | p_count |
|---|---|---|---|
| 11/23/2018 00:00:00 | 00234f1417b5b1dc6... | DIAD4VTRTSM2F4YR | 146 |
| 11/23/2018 00:00:05 | 00234f1417b5b1dc6... | INKDYVCR6P5Y9JDF | 116 |
| 11/23/2018 00:00:10 | 0024c8c4636f633a9... | RTRDFFYGTZUBKJTS | 145 |

Figure 1: Streaming Order dataset, which comes from the customer over the e-commerce website

Figure 1 is the snap of the Order dataset which comprises of 4 different fields *bought_timestamp* which represents the time when customer places the order, *uid* represents the User Identification number which is unique for each user, *pid* field represents the Product Identification number which is again unique to each product through which they are identified and *p_count* represents the count of the products ordered by the customer. The row 2,3,4 are some of the records in available in the order dataset. This order dataset is the combination of user and product dataset shown below.

| pid | registration_timestamp | product_name | retail_price | discounted_price | stock |
|---|---|---|---|---|---|
| DIAD4VTRTSM2F4YR | 2015-12-31 09:19:... | Designwallas Fest... | 590 | 590 | 161 |
| INKDYVCR6P5Y9JDF | 2016-01-03 09:13:... | Samsung CLP K300A... | 3999 | 3999 | 183 |
| RTRDFFYGTZUBKJTS | 2015-12-01 06:13:... | Belkin Play Max M... | 12499 | 12499 | 159 |

Figure 2: Product dataset, which holds information about product

The above Figure 2 is a snap of the actual Product dataset that is used. The dataset comprises are various fields each of which is used to define the product ordered by the customer. The field *pid* here represents the product identification and acts as the primary key between order and product dataset, the *registration_timestamp* represents the time at which the third party vendor registers the product for endorsement in the e-commerce portal, *product_name* represents the name of the product, *retail_price* is the actual amount of the product that the user selects, *discounted_price* is the price that will be detected from the original retail price and *stock* represents number of stocks available in the inventory of that particular product.

| uid | name | address | postcode | state |
|---|---|---|---|---|
| 63c19e1306bac1e69... | Anchor Inn | Upper Street, Str... | CO7 6LW | Babergh |
| 31c41c49376cceeb6... | Angel Inn | Egremont Street, ... | CO10 7SA | Babergh |
| 0c4e7646251e7646e... | Black Boy Hotel | 7 Market Hill, SU... | CO10 2EA | Babergh |

Figure 3: User dataset, which contains customer information

Above is a snapshot of the user dataset. The dataset contains five different fields and each of those represents customer information. The field *uid* represents the User Identification number which act as a primary key between order and user dataset. The *name* represents the customer name who places the order in the application, *address* contains the address to which the product needs to be delivered, *postcode* and *state* is the subset of the address field which is used to deliver the product.

# 5. Spark Techniques

## 5.1 Spark Streaming

Spark Streaming component supports stream processing. It is highly scalable and fault tolerant. Spark Streaming supports data from various sources such as files, sockets, Kafka, Flume, HDFS and Kinesis. It supports both batch and streaming workloads. Data streamed and processed through Spark DStreaming can be pushed out to files, HDFS, Kafka, etc.

### 5.1.1 Spark Structured Streaming

Spark Structured Streaming is a component of Spark that supports Streaming. Spark Structured Streaming accepts input from sources such as file, socket, Kafka, etc. and writes output to file, Kafka,

console, etc. It provides additional benefits such as event time, processing time, window operations, watermarking, etc.

Core data structure used here is dataframe. Event time talks about time that an event takes place. This time relies on the data and not on any external factor. Processing time is the time at which system receives the data. Window operations allow us to divide batches of data into intervals. Window operations can handle late data too. Watermarking helps to set a limit within which late data will be allowed entry into the system [3].

### 5.1.2 Spark DStreaming

DStream stands for Discretized stream. Data is broken down into batches and then processed. Batches of data is treated as RDD and RDD operations can be applied on data. The results are pushed out again in the form of batches. Batch size can lie anywhere between 1s to multiple minutes [1].

### 5.2 Spark Dataframe

Spark Dataframe is a data structure that can hold together data coming from different data types. Dataframe is like a schema which contains column title and the records. Different operations can also be executed on the dataframe to yield effective results.

### 5.3 Spark SQL

The power of SQL is known to all. Spark taps in this power of SQL within itself so that query statements can be executed to retrieve information from various relational tables and databases.

Regular SQL operations such as Create, Select, Insert, Update, Delete and Join are supported by Spark SQL [4].

# 6. Methodology

## 6.1 System Overview

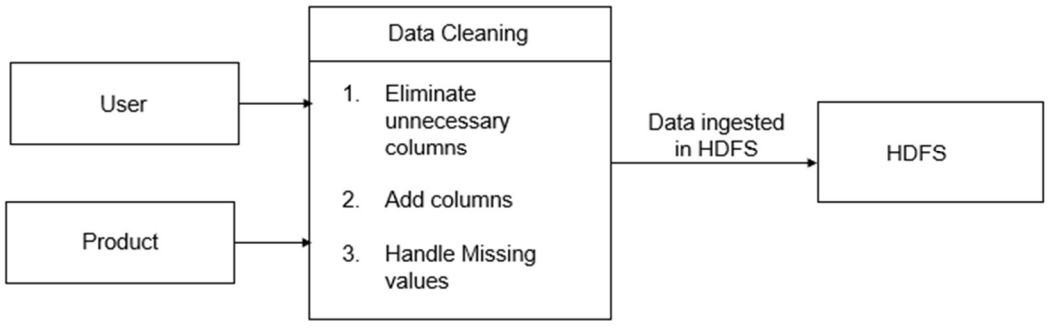



Figure 4: User and Product Dataframes cleaned and stored in HDFS

Figure 4 above shows that two datasets '*product*' and '*user*' fetched from the public domain will be prepped to make it ready for being used by our application. Data preparation would involve eliminating unused attributes, retain attributes that will prove useful in our project and making the attributes that are common between both the datasets consistent. Order dataset is created using the Product and User data set.

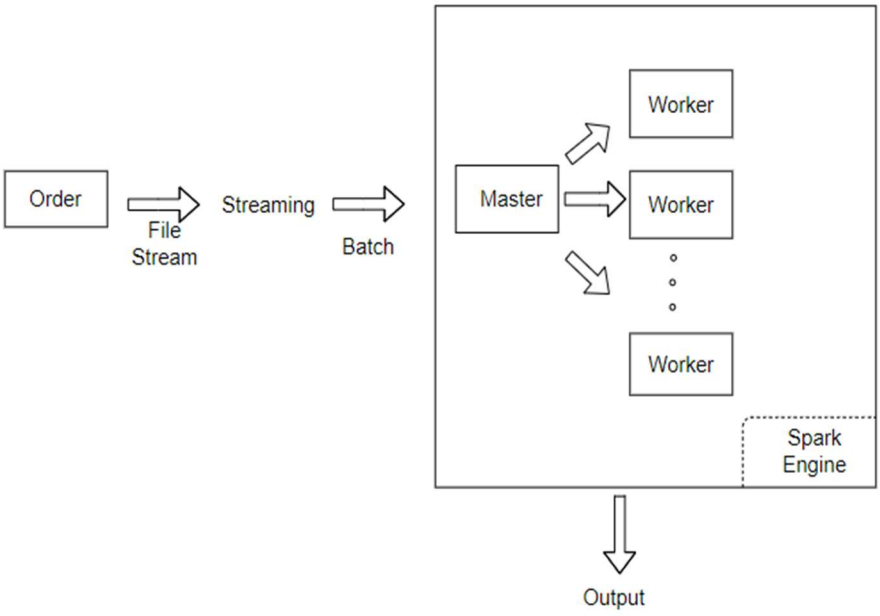



Figure 5: Order Dataframe Workflow using Spark Streaming

Figure 5 above briefly illustrates what we are trying to achieve in our project implementation. Order dataset is a simulated real-time dataset. It resembles how an Order dataset would look like. It has attributes such as an event timestamp, Product ID, User ID, etc. We use file stream to stream the order dataset. This dataset will then be used to see whether enough quantity of products is available, and the inventory will be updated. If the order can be fulfilled, then a column 'is_fulfilled' will be set to 1 or else set to 0. Based on the value of this flag the customers would understand if their products (they have ordered) will be delivered to them or not [1][3].

## 6.2 Process Flow

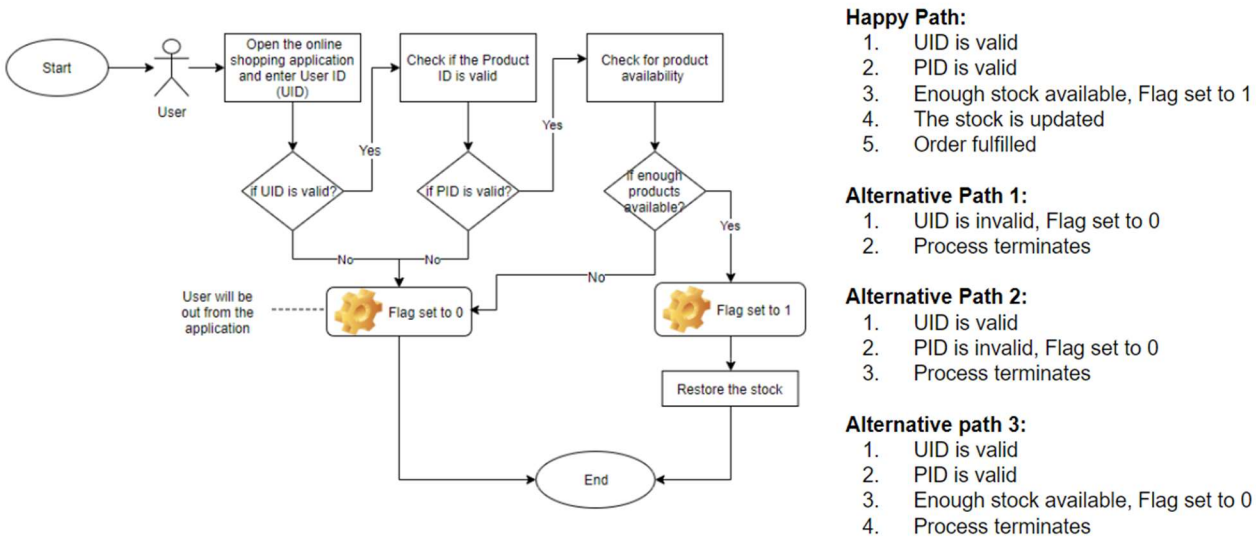



Figure 6: System process flow showing possible scenarios

The above process flow shows different scenarios that our application can run into. Scenario 1 has been termed as happy path as we are assuming that all the entered parameters for order, User and Product have passed all the gates for validation successfully. We assume that stock is available and hence the order is fulfilled.

In scenario 2, User ID present in the order does not match with any User ID data that we have in our system. This causes the process flow to be terminated and failure in order fulfillment.

In scenario 3, User ID is correct, but the product ID is not the same as that which our application Product dataset has. As a result, the order is not fulfilled.

In scenario 4, even though both user ID and Product ID are correct (User ID and Product ID matches with the User ID and Product ID that our system contains); the essential part of stock availability condition fails. This results in failure of order fulfillment.

## 6.3 Implementation

The raw datasets namely User and Product datasets (as shown above in the Dataset section) had to be cleansed to be fit for Streaming and other operations. Inherent noise present in the data must be filtered out from the datasets. Once these datasets were clean, they had to be ingested into Spark Cluster through HDFS.

Timestamps were added to the Order dataset to indicate when the order was placed (event time). Order dataset is then streamed. Spark code is executed to run the functionality that validates the stock in the inventory. This functionality is called to maintain an updated quantity of each product item in the inventory post every order fulfillment.

Below snapshot shows how an executed order dataframe looks like. The corresponding header are bought_timestamp, uid, pid, p_count, sub and if_fulfill, which means the event timestamp, user id, product id, the number of products users purchased, stock deduction and if the order fulfilled.

| _1 | _2 | _3 | _4 | _5 | _6 |
|---|---|---|---|---|---|
| '11/23/2018 00:00... | '00234f1417b5b1dc... | 'DIAD4VTRTSM2F4YR' | '146' | 15 | 1 |
| '11/23/2018 00:00... | '00234f1417b5b1dc... | 'INKDYVCR6P5Y9JDF' | '116' | 67 | 1 |
| '11/23/2018 00:00... | '0024c8c4636f633a... | 'RTRDFFYGTZUBKJTS' | '145' | 14 | 1 |
| '11/23/2018 00:00... | '0024c8c4636f633a... | 'RTRD6FYYWFHEYJYE' | '120' | 11 | 1 |
| '11/23/2018 00:00... | '002810df1681c552... | 'RTRD38494HWZVMAK' | '112' | 64 | 1 |
| '11/23/2018 00:00... | '002810df1681c552... | 'RTRDFFYGJ4GNRSHQ' | '143' | -23 | 0 |
| '11/23/2018 00:00... | '002cae16ffce9919... | 'RTRDFFYGKJAXPUUG' | '115' | 2 | 1 |
| '11/23/2018 00:00... | '002cae16ffce9919... | 'RTRDFFYGM2TEWVCC' | '148' | -48 | 0 |
| '11/23/2018 00:00... | '00371f6f8a517e4b... | 'RTRD7HN3JJYF6WN2' | '149' | -16 | 0 |
| '11/23/2018 00:00... | '00371f6f8a517e4b... | 'ABQEJ7YQTNQGMXZV' | '119' | 65 | 1 |
| '11/23/2018 00:00... | '003aebcdcfd3d30b... | 'AFGEFM64DHDFU7A4' | '149' | 31 | 1 |
| '11/23/2018 00:00... | '003aebcdcfd3d30b... | 'AFGEFM63KZJ577GG' | '143' | 21 | 1 |
| '11/23/2018 00:01... | '003fccc2bd5cfb83... | 'AFGEGF4DTH3GCFYD' | '144' | -44 | 0 |
| '11/23/2018 00:01... | '003fccc2bd5cfb83... | 'AFGEFJSDPDENNGTZ' | '112' | 24 | 1 |
| '11/23/2018 00:01... | '00610b5557d45836... | 'AFGEFM63YQH4WHRH' | '119' | -4 | 0 |
| '11/23/2018 00:01... | '00610b5557d45836... | 'AFGEFM64Z5PHJHYR' | '118' | 60 | 1 |
| '11/23/2018 00:01... | '0072836d8c9195a0... | 'AFGEFM64DHQYDJPT' | '119' | 40 | 1 |
| '11/23/2018 00:01... | '0072836d8c9195a0... | 'AFGEFJSH59XSEMVH' | '139' | -31 | 0 |
| '11/23/2018 00:01... | '00863fe42684b2bf... | 'AFGEFJSNF4QCGMXZ' | '114' | 33 | 1 |
| '11/23/2018 00:01... | '00863fe42684b2bf... | 'AFGEFJSDHHYRHQ25' | '114' | 85 | 1 |

Figure 7: Output of Streaming Order Dataframe

Order dataframe is executed on an hourly basis. After we run the program there will be a change in the stock values in the product dataframe. Below we have attached snapshots of product dataframe before and after running the program. Please note the change in the values of stock.

Original Product Dataframe

| pid | registration_timestamp | product_name | retail_price | discounted_price | stock |
|---|---|---|---|---|---|
| DIAD4VTRTSM2F4YR | 2015-12-31 09:19:... | Designwallas Fest... | 590 | 590 | 161 |
| INKDYVCR6P5Y9JDF | 2016-01-03 09:13:... | Samsung CLP K300A... | 3999 | 3999 | 183 |
| RTRDFFYGTZUBKJTS | 2015-12-01 06:13:... | Belkin Play Max M... | 12499 | 12499 | 159 |
| RTRD6FYYWFHEYJYE | 2015-12-01 06:13:... | Netgear N300 Wire... | 6400 | 5450 | 131 |
| RTRD38494HWZVMAK | 2015-12-01 06:13:... | Asus RT-N13U B1 W... | 2900 | 2100 | 176 |
| RTRDFFYGJ4GNRSHQ | 2015-12-01 06:13:... | Netgear WNDR3700 ... | 9000 | 6300 | 120 |
| RTRDFFYGKJAXPUUG | 2015-12-01 06:13:... | Belkin Basic Surf... | 3099 | 3099 | 117 |
| RTRDFFYGM2TEWVCC | 2015-12-01 06:13:... | Belkin Share Mode... | 6249 | 6249 | 100 |
| RTRD7HN3JJYF6WN2 | 2015-12-01 06:13:... | TP-LINK TL-WR841N... | 2500 | 1118 | 133 |
| ABQEJ7YQTNQGMXZV | 2016-06-12 08:33:... | Momin Libas AK241... | 3999 | 2398 | 184 |
| AFGEFM64DHDFU7A4 | 2016-03-03 06:06:... | Great Eastern Sou... | 3276 | 2784 | 180 |
| AFGEFM63KZJ577GG | 2016-03-03 06:06:... | Moulin Roty Les P... | 4122 | 3503 | 164 |
| AFGEGF4DTH3GCFYD | 2016-03-03 06:06:... | Wild Republic Nat... | 549 | 549 | 100 |
| AFGEFJSDPDENNGTZ | 2016-03-03 06:06:... | Nendoroid Wf2014W... | 8509 | 7232 | 136 |
| AFGEFM63YQH4WHRH | 2016-03-03 06:06:... | Discovery Toys Ma... | 11435 | 9719 | 115 |
| AFGEFM64Z5PHJHYR | 2016-03-03 06:06:... | Tom and Jerry 9 P... | 4141 | 3519 | 178 |
| AFGEFM64DHQYDJPT | 2016-03-03 06:06:... | Beyblade Metal Fu... | 10416 | 8853 | 159 |
| AFGEFJSH59XSEMVH | 2016-03-03 06:06:... | "Sentinel Braveac... | 11431 | 9716 | 108 |
| AFGEFJSNF4QCGMXZ | 2016-03-03 06:06:... | Sentinel Mak Furi... | 19345 | 16443 | 147 |
| AFGEFJSDHHYRHQ25 | 2016-03-03 06:06:... | Nendoroid Vocalol... | 8757 | 7443 | 199 |

Updated Product Dataframe

| pid | registration_timestamp | product_name | retail_price | discounted_price | stock |
|---|---|---|---|---|---|
| DIAD4VTRTSM2F4YR | 2015-12-31 09:19:... | Designwallas Fest... | 590 | 590 | 15 |
| INKDYVCR6P5Y9JDF | 2016-01-03 09:13:... | Samsung CLP K300A... | 3999 | 3999 | 67 |
| RTRDFFYGTZUBKJTS | 2015-12-01 06:13:... | Belkin Play Max M... | 12499 | 12499 | 14 |
| RTRD6FYYWFHEYJYE | 2015-12-01 06:13:... | Netgear N300 Wire... | 6400 | 5450 | 11 |
| RTRD38494HWZVMAK | 2015-12-01 06:13:... | Asus RT-N13U B1 W... | 2900 | 2100 | 64 |
| RTRDFFYGJ4GNRSHQ | 2015-12-01 06:13:... | Netgear WNDR3700 ... | 9000 | 6300 | 120 |
| RTRDFFYGKJAXPUUG | 2015-12-01 06:13:... | Belkin Basic Surf... | 3099 | 3099 | 2 |
| RTRDFFYGM2TEWVCC | 2015-12-01 06:13:... | Belkin Share Mode... | 6249 | 6249 | 100 |
| RTRD7HN3JJYF6WN2 | 2015-12-01 06:13:... | TP-LINK TL-WR841N... | 2500 | 1118 | 133 |
| ABQEJ7YQTNQGMXZV | 2016-06-12 08:33:... | Momin Libas AK241... | 3999 | 2398 | 65 |
| AFGEFM64DHDFU7A4 | 2016-03-03 06:06:... | Great Eastern Sou... | 3276 | 2784 | 31 |
| AFGEFM63KZJ577GG | 2016-03-03 06:06:... | Moulin Roty Les P... | 4122 | 3503 | 21 |
| AFGEGF4DTH3GCFYD | 2016-03-03 06:06:... | Wild Republic Nat... | 549 | 549 | 100 |
| AFGEFJSDPDENNGTZ | 2016-03-03 06:06:... | Nendoroid Wf2014W... | 8509 | 7232 | 24 |
| AFGEFM63YQH4WHRH | 2016-03-03 06:06:... | Discovery Toys Ma... | 11435 | 9719 | 115 |
| AFGEFM64Z5PHJHYR | 2016-03-03 06:06:... | Tom and Jerry 9 P... | 4141 | 3519 | 60 |
| AFGEFM64DHQYDJPT | 2016-03-03 06:06:... | Beyblade Metal Fu... | 10416 | 8853 | 40 |
| AFGEFJSH59XSEMVH | 2016-03-03 06:06:... | "Sentinel Braveac... | 11431 | 9716 | 108 |
| AFGEFJSNF4QCGMXZ | 2016-03-03 06:06:... | Sentinel Mak Furi... | 19345 | 16443 | 33 |
| AFGEFJSDHHYRHQ25 | 2016-03-03 06:06:... | Nendoroid Vocalol... | 8757 | 7443 | 85 |

Figure 8: Product Dataframe Before and After Stock Update

The above Figure 8 shows the difference between the original product dataframe that is used vs the updated product dataframe that is configured after screaming is performed. The major difference between the two images are the stock column.

# 7. Evaluation

## 7.1 Big Data Cluster Description

Worker nodes perform the major computational operations on the cluster. Each node has two 18-core Intel Xeon Gold 6140 Skylake CPUs (2.3 GHz clock speed, 24.75 MB L3 cache, 6 memory channels, 140 W power), for a total of 36 cores per node. Each node has 384 GB of memory (12 x 32 GB DDR4 at 2666 MT/s) and 48 TB (12 x 4 TB) SATA hard disks, and each node is connected to each other node by a 10 Gb/s Ethernet network. These 8 nodes, in addition to two other nodes as well as ancillary hardware that facilitates the network, compose the entirety of the Big Data Cluster.

## 7.2 Experiment

We attempted to measure the latency and execution time when we dealt with orders of different data size. We have a configuration of one executor and one core. When we executed the program, we noticed that there is an upward trend in time when there is a rise in the number of records in the order dataframe.

Scatter plot below shows the time taken to execute order dataframes of different sizes.

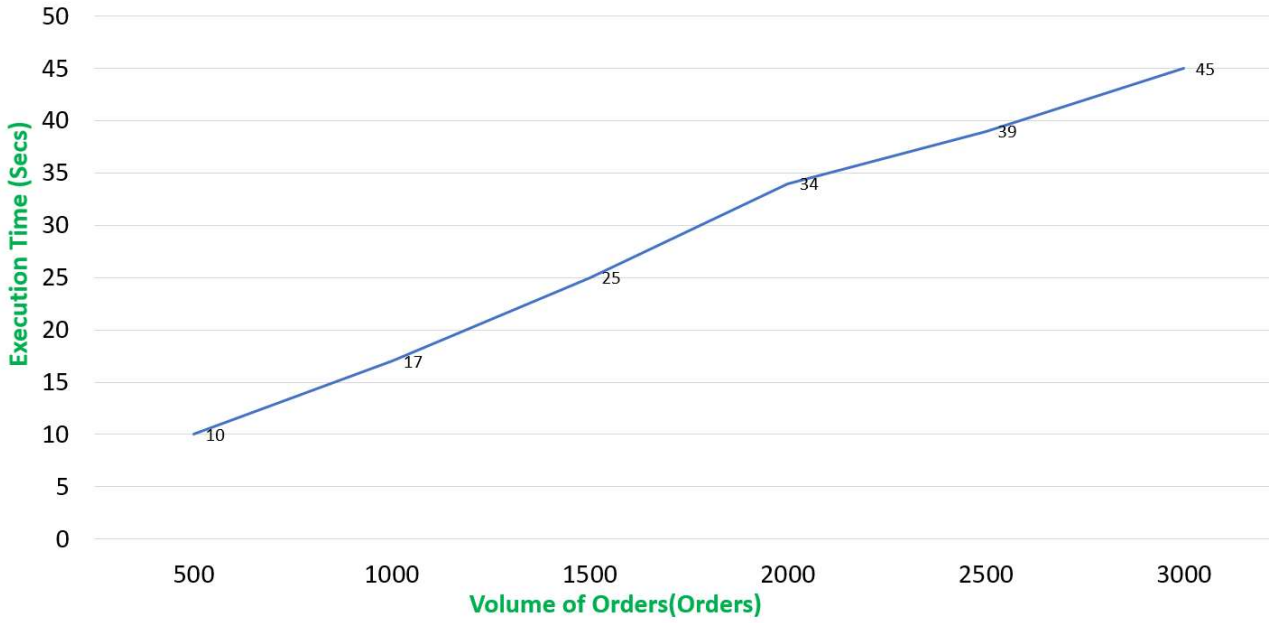



Figure 9: Volume Measures indicate rise in time with rise in orders

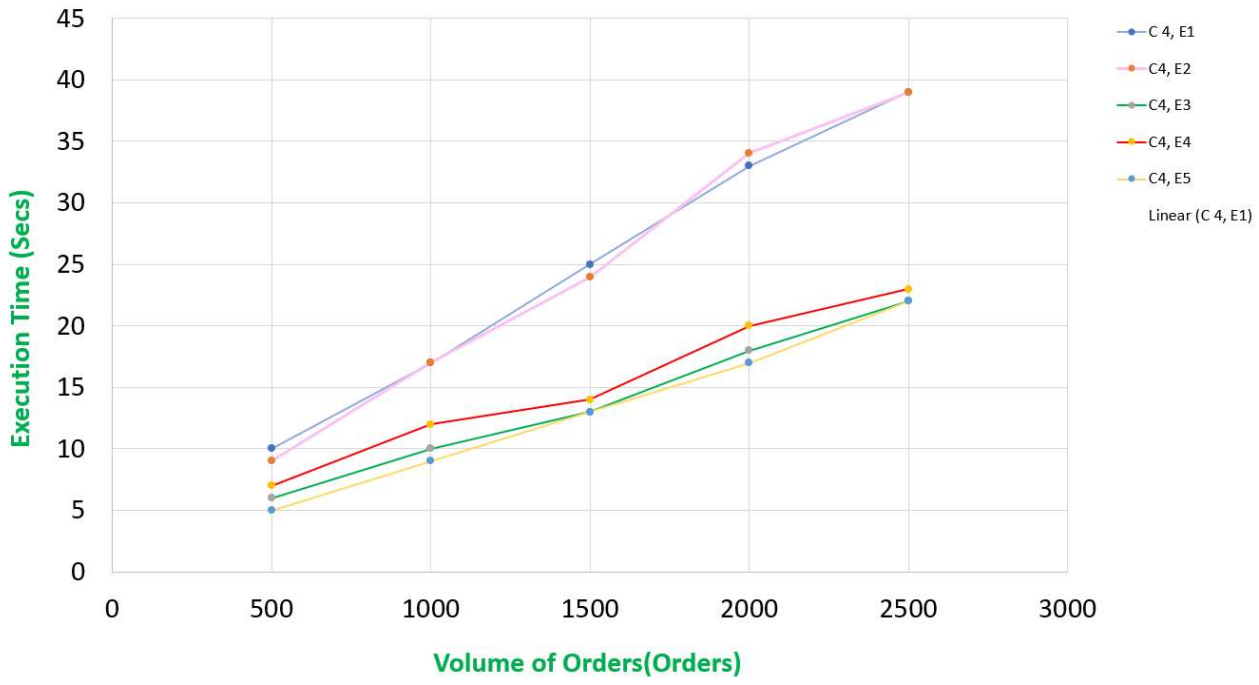



Figure 10: Scalability Measures showing orders volume vs Execution time for different configurations of executors and core

Figure 9 shows that execution time depicts a rising trend when there is a rise in the number of orders. Figure 10 shows the scalability, task distribution and time taken for our application. Scatter plot was used for visualization that shows the time consumed for a set of orders in figures above. In Figure 10, different combinations of cores and executors were used. One of the configurations was E1- 1 Executor and C4- 4 Cores were used. A legend on the top-right of Figure 10 shows the various configurations for which we have measured the time.

We carried out an experiment to address scalability of the application developed. Five configurations were used for the experiment; each configuration had a fixed set of four cores and rising number of executors starting with one executor and continuing to five executors. The only noticeable observation we found was that as the volume of orders increased, the execution time also increased in parallel. In general, it was evident that more executors consumed less time. So far, E5(5 executors) had the best performance as it took shortest execution time. In some cases, when more executors were added there was a rise in time as opposed to task completion in less time. This might have been due to network congestion or cluster environment being used by multiple people.

Memory and a greater number of cores when added for execution did not significantly affect the execution time. Each executor has a core associated with it. We made note of one more thing, when we carried out execution using four executors, the tasks were distributed amongst the executors, so each executor will deal with parts of tasks. The total time taken by the cluster for completing the program will decrease. That means when executors are more in number, then the efficiency of the program increases as distribution of task takes place amongst the executors. This configuration would help solve efficiency problems for the e-commerce companies to handle orders traffic; as executors work in parallel and the tasks will be distributed.

| Executor ID | Address | Status | RDD Blocks | Storage Memory | Disk Used | Cores | Active Tasks | Failed Tasks | Complete Tasks | Total Tasks | Task Time (GC Time) | Input | Shuffle Read | Shuffle Write |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| driver | login.hdp-internal:35948 | Active | 0 | 0.0 B / 384.1 MB | 0.0 B | 0 | 0 | 0 | 0 | 0 | 0 ms (0 ms) | 0.0 B | 0.0 B | 0.0 B |
| 1 | worker8.hdp-internal:36375 | Active | 0 | 0.0 B / 428.8 MB | 0.0 B | 1 | 0 | 0 | 4 | 4 | 45 s (0.1 s) | 2.1 MB | 0.0 B | 0.0 B |

Figure 11: Execution time for 1 core is 45 seconds

| Executor ID | Address | Status | RDD Blocks | Storage Memory | Disk Used | Cores | Active Tasks | Failed Tasks | Complete Tasks | Total Tasks | Task Time (GC Time) | Input | Shuffle Read | Shuffle Write |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| driver | login.hdp-internal:36697 | Active | 0 | 0.0 B / 384.1 MB | 0.0 B | 0 | 0 | 0 | 0 | 0 | 0 ms (0 ms) | 0.0 B | 0.0 B | 0.0 B |
| 1 | worker1.hdp-internal:40782 | Active | 0 | 0.0 B / 428.8 MB | 0.0 B | 1 | 0 | 0 | 2 | 2 | 18 s (0.1 s) | 1.4 MB | 0.0 B | 0.0 B |
| 2 | worker1.hdp-internal:39026 | Active | 0 | 0.0 B / 428.8 MB | 0.0 B | 1 | 0 | 0 | 0 | 0 | 0 ms (0 ms) | 0.0 B | 0.0 B | 0.0 B |
| 3 | worker1.hdp-internal:40989 | Active | 0 | 0.0 B / 428.8 MB | 0.0 B | 1 | 0 | 0 | 2 | 2 | 28 s (86 ms) | 711.9 KB | 0.0 B | 0.0 B |
| 4 | worker1.hdp-internal:45227 | Active | 0 | 0.0 B / 428.8 MB | 0.0 B | 1 | 0 | 0 | 0 | 0 | 0 ms (0 ms) | 0.0 B | 0.0 B | 0.0 B |

Figure 12: Task Distribution among multiple parallel Spark Executors via Spark UI

In Figure 12, execution time for 4 cores and 4 executors is 45 seconds (sum of execution time of all executors); largest time taken by one executor is 28 seconds.

## 8. Conclusion

This application helped us understand that Spark is very powerful as it helps in processing large datasets. Streaming is very useful to simulate real-world scenarios. Spark SQL is extremely beneficial to retrieve specific data applicable for a functionality.

Please find the project details from the link: https://github.com/starlyxxx/spark_bigdata

## 9. Future Work

There is scope to learn about how we can handle multiple query processing simultaneously using Spark Structured Streaming. In our future endeavors, we can also work to improve performance, reduce latency and throughput to increase the efficiency of the application. We must try to maintain the network availability for smooth operation of the application. Accuracy of stock needs to be maintained always to ensure faster rate of customer order fulfillment.

## Acknowledgement

The hardware in the UMBC High Performance Computing Facility (HPCF) is supported by the U.S. National Science Foundation through the MRI program (grant nos. CNS–0821258, CNS–1228778, and OAC–1726023) and the SCREMS program (grant no. DMS–0821311), with additional substantial support from the University of Maryland, Baltimore County (UMBC). We would like to extend special thanks to our instructor Dr. Jianwu Wang for supporting us.

## References


[1]. Armbrust, M., Das, T., Torres, J., Yavuz, B., Zhu, S., Xin, R., ... & Zaharia, M. (2018, May). Structured streaming: A declarative API for real-time applications in apache spark. In Proceedings of the 2018 International Conference on Management of Data (pp. 601-613). ACM.

[2]. Shoro, A. G., & Soomro, T. R. (2015). Big data analysis: Apache spark perspective. Global Journal of Computer Science and Technology.

[3]. Ivanov, T., & Taaffe, J. (2018, April). Exploratory Analysis of Spark Structured Streaming. In Companion of the 2018 ACM/SPEC International Conference on Performance Engineering (pp. 141-146). ACM.

[4]. Michael Armbrust, Reynold S. Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K. Bradley, Xiangrui Meng, Tomer Kaftan, Michael J. Franklin, Ali Ghodsi, Matei Zaharia (2015). Spark SQL: Relational Data Processing in Spark

[5]. Salloum, S., Dautov, R., Chen, X., Peng, P. X., & Huang, J. Z. (2016). Big data analytics on Apache Spark. International Journal of Data Science and Analytics, 1(3-4), 145-164.

[6]. Chand, M., Shakya, C., Saggu, G. S., Saha, D., Shreshtha, I. K., & Saxena, A. (2017). Analysis of big data using apache spark. In 4th International conference on computing for sustainable global development. BVICAM.

[7]. Shanahan, J. G., & Dai, L. (2015, August). Large scale distributed data science using apache spark. In Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining (pp. 2323-2324). ACM.

[8]. Jonnalagadda, V. S., & Thumati, K. (2016). A review study of apache spark in big data processing. International Journal of Computer Science Trends and Technology (IJCST), 4(3), 93-98.

[9]. Bansal, A., Jain, R., & Modi, K. (2019). Big Data Streaming with Spark. In Big Data Processing Using Spark in Cloud (pp. 23-50). Springer, Singapore.

[10]. Chellappan, S., & Ganesan, D. (2018). Spark Structured Streaming. In Practical Apache Spark (pp. 157-174). Apress, Berkeley, CA.