Varsini Dhinakaran (vd299) Sahmi Abubakar (saa315)
Mehraaj Tawa (mt1055) Sohum Pohane (sdp195)

Cloud Computing Final Project Report

***Note**: This report is aimed to supplement the in person presentation. Example plots and code will be submitted or shown through the presentation*

Background Information:

Apache spark is an open source distributed computing system that allows for fast, general purpose computer framework for high volume data processing. It has several key functionalities that allow it to be highly suited towards various data processing and analytics tasks. For one, spark functions through distributed processing of large data sets across a cluster of machines. These datasets are divided into partitions and then processed in parallel, allowing its functions to be easily scaled while still being efficient.

The main data structure used in spark is a resilient distributed dataset (RDD), which represent immutable distributed collections of objects that are able to be processed in parallel. RDDs are fault-tolerant, which means that they can recover from node failure during computation. This is important as the ability to recover quickly from hardware failure, software errors or other unexpected issues is integral for ensuring the reliability and integrity of distributed systems.

Using RDDs, various operations like map, filter, reduce and join are used to process, manipulate and transform data to suit the user's needs. These functions are created in a way to facilitate the distributed nature of the work environment. Moving past some basic functionalities, spark also facilitates other types of applications like SQL, Machine Learning or Graph Processing. Spark SQL provides an interface for data manipulation that closely resembles SQL queries. MLlib is the scalable machine learning library included with spark that provides algorithms for classification, regression and filtering for machine learning applications. The last functionality to be highlighted is GraphX, the graph processing library included that enables processing of graph-structured data.

The second technological component of the project is pandas, another open source data manipulation and analysis library for Python. The key difference between Spark and pandas is the nature of processing between the two. Pandas is distinctly local while spark allows for distributed computation, making pandas the slower option for high volume data processing. However, pandas' functionality and ability to visualize data through other libraries like SeaBorn make it a useful tool for data processing.

Pandas also uses a table-like data structure, known as a data frame. Methods like data reshaping, pivoting and melting can be performed on a data frame, in addition to functions that handle missing data. All of these available methods allow for ease of data processing and filtering, and when incorporated with visualization libraries, create a fleshed out end product for the user.

It's only drawback is its processing scope, as it is a local processing program. However, with the combined use of Spark, the processing power is greatly increased, allowing for efficient computation and while visual integration is provided through pandas.

Spark already has a pandas API, however it is still limited to work in a local setting. All spark partitions of the RDD will be sent to one worker, often causing a memory overflow. But, as of September 13th, 2023 (just two months ago), pySpark 3.5 came out with its own built-in support for pandas in a distributed fashion, efficiently taking advantage of the strengths of both technologies.

Our Project:

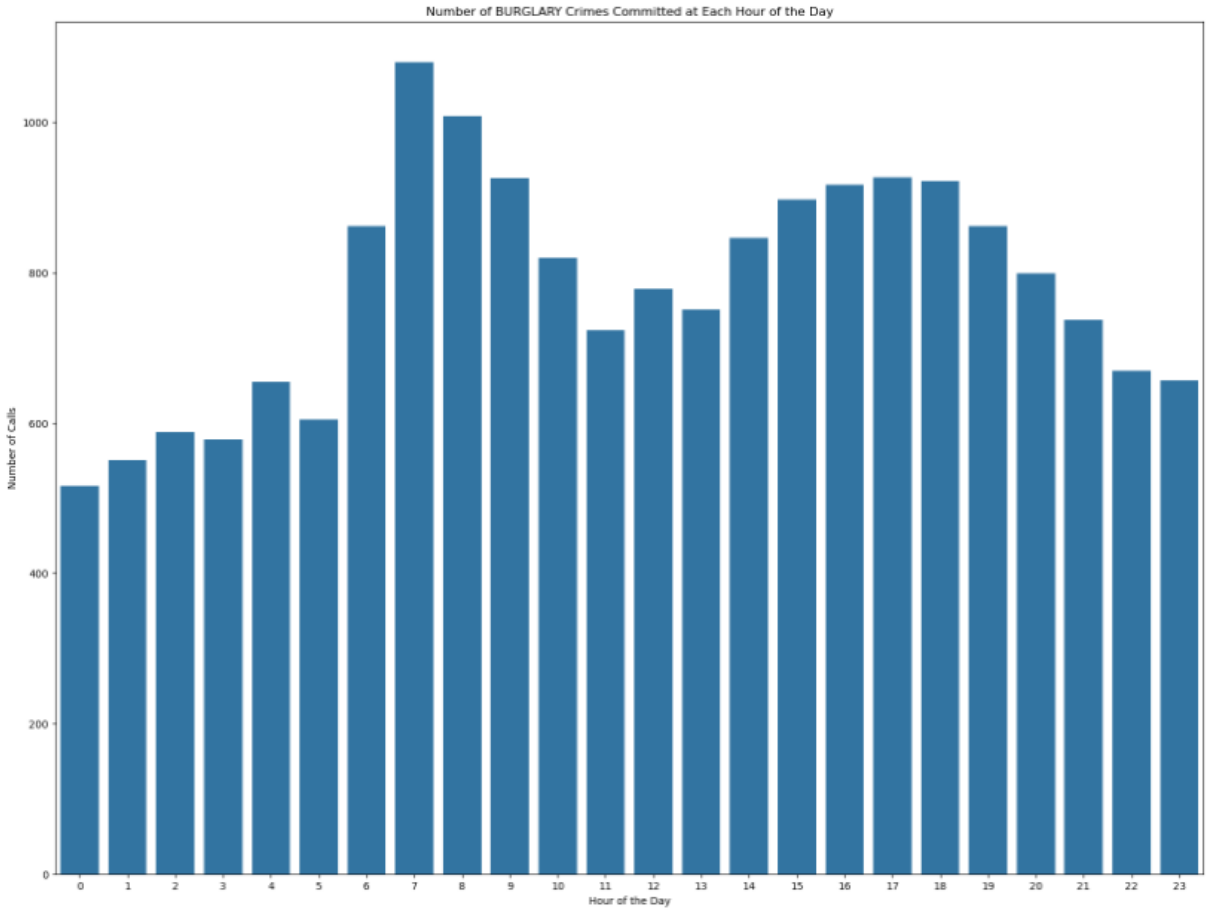
The aim of this project is to take advantage of this new pandas-over-spark API by providing a cohesive view of all crime statistics in a given area through the use of larger datasets that will be preprocessed and feature engineered in a distributed fashion. The motivation for this project stems from the need to know crime behavior in states, as graduating seniors are moving to accommodate for their work opportunities. As such, it is important to know if the area is safe or safe enough when considering areas for relocation.

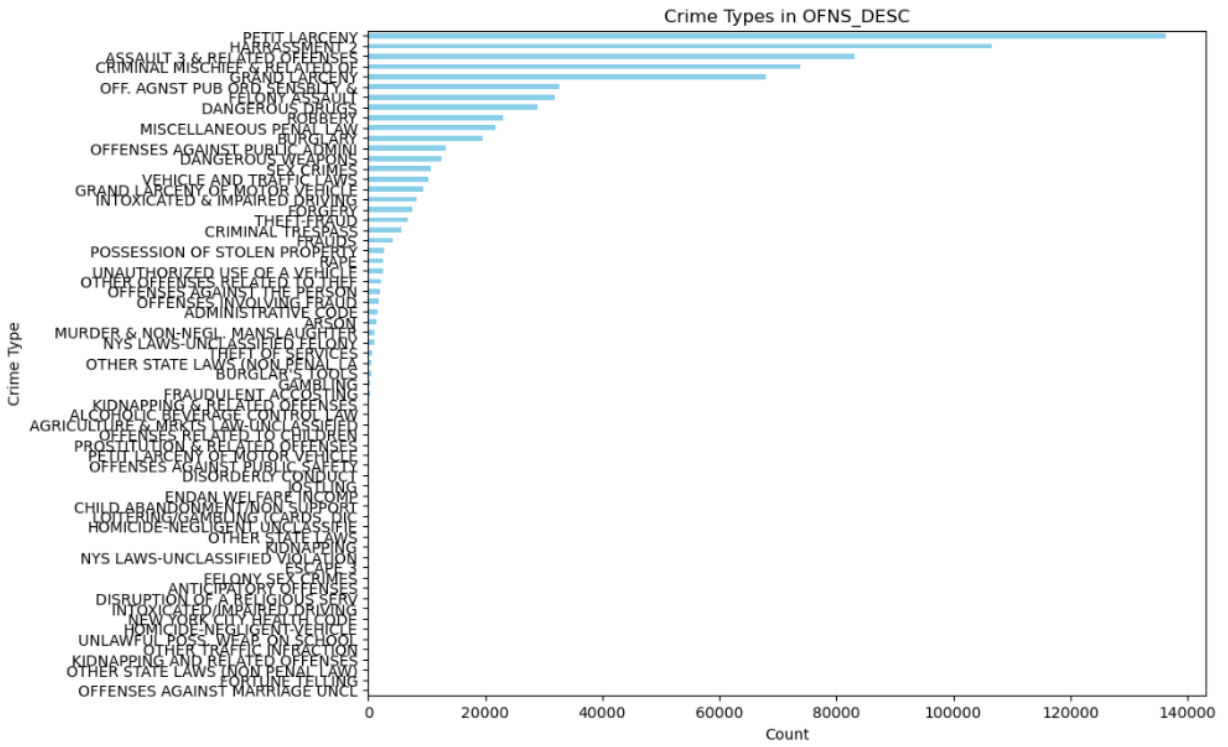
For the project, we will be using New York City's crime statistics, publicly available through the NYPD. This dataset includes date, time, location, type of crime, if an arrest was made and the crime was armed. Using this dataset, we compile different statistics to display to the user. Some of these statistics include the total number of arrests made for a given offense type, which hour of the day certain crimes occur, and a showcase of most popular crime areas onto a NY map. However, more can be added if needed easily through the same filtering and manipulation techniques used to provide the other statistics. In this way, the program can be customized in the future to provide necessary statistics based on either the area or just the desired statistics of the user.

These stats are found by isolating the field required and operating on only those rows or columns. This data is then fed to SeaBorn which is a visualization library that takes the data and outputs a graph that has been defined. For example, when finding the most popular type of crime, the data is grouped by the type of crime and then counted to see which is the most prevalent in the area. The result is given to SeaBorn which outputs an ordered horizontal bar graph allowing for the user to easily recognize the most popular crimes in the area. A few examples of plots that we can generate are shown below:

Distribution of Hour of the Day by Crime Category
Hour of the Day
0
5
10
15
20
DANGEROUS WEAPONS
FORGERY
HARRASSMENT 2
MISCELLANEOUS PENAL LAW
MURDER & NON-NEGL. MANSLAUGHTER
BURGLARY
DANGEROUS DRUGS
PETIT LARCENY
OFF. AGNST PUB ORD SENSBLTY &
GRAND LARCENY
FELONY ASSAULT
ASSAULT 3 & RELATED OFFENSES
ARSON
RAPE
SEX CRIMES
GRAND LARCENY OF MOTOR VEHICLE
ROBBERY
CRIMINAL MISCHIEF & RELATED OF
THEFT FRAUD
VEHICLE AND TRAFFIC LAWS
CRIMINAL TRESPASS
OFFENSES INVOLVING FRAUD
FRAUDS
OFFENSES AGAINST PUBLIC ADMINI
OFFENSES AGAINST THE PERSON
ADMINISTRATIVE CODE
INTOXICATED & IMPAIRED DRIVING
ESCAPE 3
NYS LAWS-UNCLASSIFIED FELONY
POSSESSION OF STOLEN PROPERTY
THEFT OF SERVICES
KIDNAPPING & RELATED OFFENSES
OTHER OFFENSES RELATED TO THEF
UNAUTHORIZED USE OF A VEHICLE
BURGLAR'S TOOLS
ENDAN WELFARE INCOMP
FRAUDULENT ACCOSTING
AGRICULTURE & MRKTS LAW UNCLASSIFIED
OTHER STATE LAWS (NON PENAL LA
OFFENSES AGAINST PUBLIC SAFETY
GAMBLING
PETIT LARCENY OF MOTOR VEHICLE
ALCOHOLIC BEVERAGE CONTROL LAW
OFFENSES RELATED TO CHILDREN
ANTICIPATORY OFFENSES
LOITERING/GAMBLING (CARDS, DIC
FELONY SEX CRIMES
HOMICIDE-NEGLIGENT,UNCLASSIFIE
PROSTITUTION & RELATED OFFENSES
JOSTLING
CHILD ABANDONMENT/ NON SUPPORT
OTHER STATE LAWS
KIDNAPPING
NYS LAWS-UNCLASSIFIED VIOLATION
DISORDERLY CONDUCT
DISRUPTION OF A RELIGIOUS SERV
OFFENSES AGAINST MARRIAGE UNCL
HOMICIDE-NEGLIGENT-VEHICLE
INTOXICATED/IMPAIRED DRIVING
KIDNAPPING AND RELATED OFFENSES
UNLAWFUL POSS. WEAP. ON SCHOOL
OTHER TRAFFIC INFRACTION
OTHER STATE LAWS (NON PENAL LAW)
FORTUNE TELLING
NEW YORK CITY HEALTH CODE
Crime Category


Wayne
Fair Lawn
Lincoln Park
Paterson
Bergenfield
Yonkers
Tuckahoe
Mamaroneck
Larchmont
New Rochelle
Bayville
Woodland Park
Hackensack
Hasbrouck Heights
Clifton
Fort Lee
Glen Cove
Oyster Bay
Manorhaven
Port Washington
Syosset
Verona
Nutley
Cliffside Park
Bloomfield
North Bergen
Great Neck
Jericho
Livingston
North Arlington
Union City
Albertson
Hicksville
East Orange
Salisbury
Bethpage
Maplewood
Newark
Garden City
Levittown
Summit
Hempstead
Union
Elmont
North Valley Stream
North Bellmore
Massapeq
Westfield
Elizabeth
Bayonne
Valley Stream
Freeport
Queens County
Oceanside
Clark
Linden
Inwood
Gateway National Recreation Area
Long Beach
Richmond County
Woodbridge Township
Leaflet | © OpenStreetMap contributors


Number of BURGLARY Crimes Committed at Each Hour of the Day
Number of Calls
Hour of the Day
0
200
400
600
800
1000
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23



What makes this project unique is its scale. Commonly, data preprocessing with pandas specifically, which again is an industry standard, is done under local settings with powerful hardware. Or datasets are broken down into chunks to operate on through frameworks that do not have as many functions or flexibility. **However now, the operations and transformation variety of pandas can happen in a parallel way allowing for speed, efficiency, increased capacity, and increased compatibility (with third party softwares) compared to base spark RDD functions across various types of preprocessing jobs for many industries, fields, or applications.** Now, spark can be even more widely used by businesses because a central database or distributed file system can host *large* datasets, requests can come in by users to process data in a certain way, the central dataset can be manipulated quickly and efficiently through pandas-over-spark and horizontal scaling of nodes, and then the results can be sent to the user. By using the API in this way, microservices or specialized modules can be developed to be a core part of certain large scale applications. All it would take is to have the underlying distributed container orchestration and environment in place. No special changes are necessary to accommodate for pandas-over-spark since it is a built-in API with pySpark 3.5. Existing pandas code can be placed into a spark environment without issues.

Challenges:

Starting off, pandas-over-spark requires pySpark 3.5. However for our projects and assignments we have been using pySpark 3.2, under a docker container environment. So, the first step was to build an image for our containers that had pySpark 3.5.

As mentioned before, pySpark 3.5 is extremely new, so it was difficult to find pre-built images on Dockerhub that we could simply pull, similar to how we did for gettyimages/spark (the image used for spark 3.2). What we had to do was build our own image which was an extensive process because of the different working environments, in terms of linux vs windows. A common problem we would run into is the differing line ending characters used by both operating systems. We would have to pull dependencies that were not easily modifiable to inform which line ending type to use since the number of files would be numerous. Moreso, the more prevalent issue is that some of the dependencies needed to build the image were only for linux users, as windows versions of the dependencies have not been updated to support pySpark 3.5.

After realizing building an image would not be suitable, we were able to find one image on Dockerhub named bitnami. This image was prebuilt and had what we were looking for, however needed some adjustments in order to get working.

The second main challenge is related to the development process. We would work with large datasets in order to prove the efficiency of pandas-over-spark. However, given that we all own standard laptop machines, creating workers would take up resources such as computer cores, memory, and disk space. This would cause our computers to run extremely slow. We shifted to working with a smaller dataset for development purposes but this still would not solve the problem fully since we would need multiple workers to fully take advantage of this API. Having only one master with no other nodes was not realistic as the standard Pandas API could be used in this setting. So, we maintained having at least one master and an additional worker, and dealt with the slow speeds related to development and troubleshooting.

Another challenge we faced over the course of the project was the learning curve. We all had to learn about pandas and common data science techniques for preprocessing and feature engineering. Specifically, how to deal with categorical variable types, non-entries, and reshaping data frames to only have relevant information. This was resolved through reading pandas documentation and getting used to another data manipulation library known as NumPy. The differences between data frames, series and numpy arrays were important distinctions to understand in order for data to be stored and plotted correctly using seaborn.

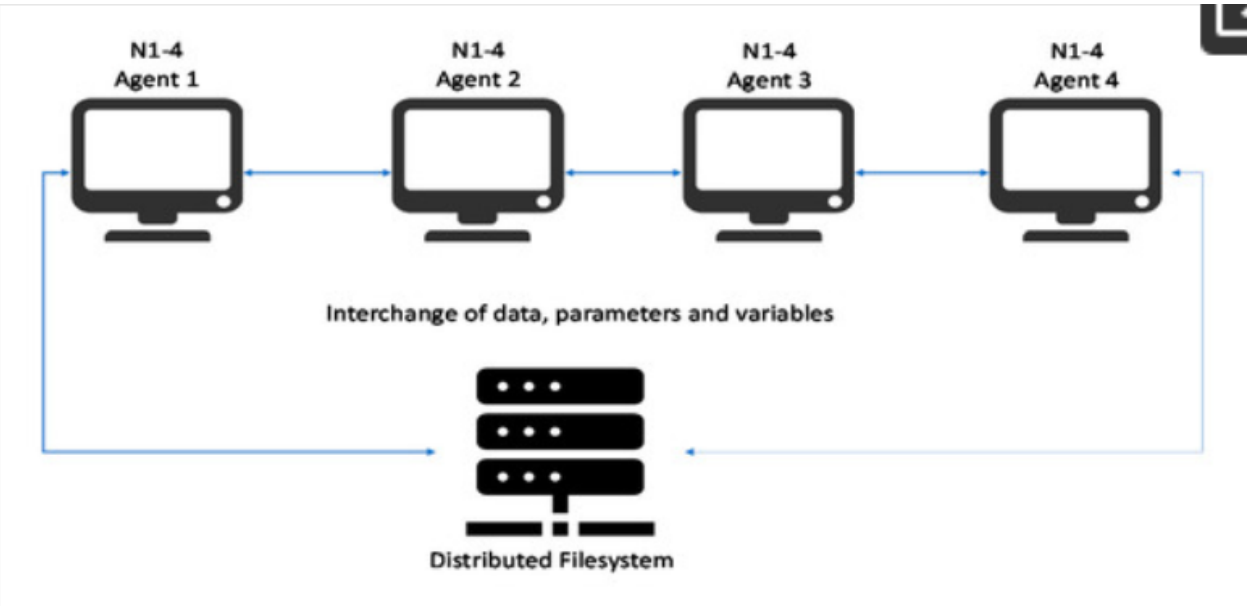
The last challenge we faced was related to data visualizations. Ideally, we were hoping to have the spark workers be able to do the data visualization. However, when we reached this part of the project we found out that spark does not handle visualizations by itself. It needs an external tool to help with the visualizations after the data is pulled from all partitions. Upon realizing this, we shifted methodologies to pull the data into csv files and then plot the data on a jupyter notebook. This change actually turned out to be a big learning experience for us because typically, this is how data plotting would be done, in the sense that spark is responsible for manipulating the data but you would need to pull the data or store the data in a centralized location so that another API or application can plot it (in a form other than just displaying the tabular data).

Applications & Workflow:

While this project currently works with one dataset from the NYPD, it can easily process other datasets from other departments, allowing the program to be used for more than New York City. If these datasets collect different information, then different statistics can efficiently be computed as well since another strength of pandas is easy exploratory data analysis (EDA). There are many possibilities of what type of information can be extracted from a large scale data set because we no longer need to extract subsets of data to represent the whole. We can simply use all of the data collected to get the most accurate representation/computation.

To elaborate on this idea further, datasets can be appended to in real time creating one central source of raw data, perhaps with additional attributes going forward that previous entries may not have. Then, when data is needed, a server can start a spark job taking advantage of an already established distributed system of containers or servers to do pandas-over-spark data preprocessing and feature engineering. Then it can return the processed data for the backend server to plot or feed into a machine learning model that the user can use. The user could request to see crime in a certain borough, counts of a certain crime, return a model that can predict New York based crime data, retrieve information about different cities, or see how often people end up reporting crime. These are only a few examples and only related to the crime reporting field.

It's important to note the true benefit of this project is to show how a microservice like this, being optimized distributed preprocessing of data, can be applied in a professional setting such as crime analysis or in other industries and fields, while also taking advantage of the ease, power and efficiency of pandas. Currently, when it comes to distributed preprocessing, some applications do the following. Nodes share a distributed file system to read from and write to. Each node is multithreaded, and each node is assigned a particular column to look at or feature to look at. Then, each thread in each node is assigned a feature to focus on to process, and then write back onto the distributed file system with locks in place to prevent overwriting and manage critical operations. If a feature warrants multiple steps to be manipulated, then each of the nodes are connected to each other in order to pass on variables, parameters or information. An image of the topology is below:



However, here every node needs to be able to read the whole dataset, which can be restricting on how large the dataset can be. Also, pandas may or may not be viable in setups like these.

In other implementations for distributed preprocessing, specific libraries like tensorFlow or pyTorch, which are at their core machine learning frameworks that only vaguely support data processing, allow for distributed preprocessing through data sharding. This means that each worker partitions and is responsible for a small section of the data, very similar to how spark works but not as optimized. However in this case, each tool has its own syntax, own backend methodologies, and limited functions available for the sake of preprocessing data since that is not the focus of their frameworks. Pandas offers more functions for a wide variety of data types, indexing preferences, data merging, time series, and more.

In our implementation, what spark is doing is partitioning sizes of the dataset to each worker. Each node can do manipulations on any column, and the data is aggregated before storing on the distributed file system or writing to the mounted drive. In this formation, the size of the dataset is not limited and through the use of pandas, a wide variety of computed statistics can reflect all of the data, not just a subset that can fit on one machine. Many types of operations can be supported, and the ease of use of pandas and its compatibility with other libraries is maintained.

So all in all, the work presented here is a proof of concept for a preprocessing module or microservice that we very well might see in the future given the release of the pandas-over-spark API. The workflow for a large scale application using this microservice could look like the following:

Frontend → backend → connect to spark environment and read distributed file system for database → compute RDD → send back to backend for visualizations or to provide to a machine learning framework→ send results to user

Shortcomings:

- Minimal documentation is an issue to understand the differences between different types of dataframes
- On top of this, speed can be an issue. Building a large dataset sounds good but only is applicable when appropriate horizontal scaling is established. Multiple workers are needed to take advantage of the parallelization. Otherwise, it might be better to stick to existing technologies.
- Networking bottlenecks in terms of the overhead from read/write from shared file system, communicating partitions to master node and sending processed data to another framework
- This solution can be over delivering in situations with multiple small files or for situations where only simple data processing is necessary.