COMP 1702
Big Data Coursework Part 2.

Student ID - 001002629

29th March 2021

Contents

1 Part A - MapReduce Programming

Algorithm 1: Algorithm for Finding Number of Papers for each Conference

```
Class Mapper:
    method Map(key k, value v)
        value v ← v delimited by |          // changing delimiter
        for term t ϵ value v do
            return (term t, count 1)

Class Reducer:
    method Reduce(term t, counts [c1, c2...])
        sum ← 0
        for term t ϵ value v do
            sum ← sum + c
        return (term t, count sum)
```

Algorithm 1 shows a basic implementation written in pseudocode that would iterate over papers.txt and sum the values that are found. Mapper class changes the delimiting value from default white space to the vertical bar that is found in the text. This changes the key from a single word to multiple words if there are multiple words present while mapping out values. As each map is a single instance a loop is present to send the mapped term and an integer to the reducer class.

The reducer class take the output from the mapper class and uses the terms and number of counts as inputs, initialising a variable to hold the count to 0 before any reductions proceed makes sure that this variable is initialised and ready to be used. The for loop that is in the reduce class then matches each unique *term* variable that is present and increments the variable *sum* for each instance found.

There is an issue with this particular implementation, that it will map and reduce all parts of the line in the text file. So it will map all authors, title, conference and year separately which is inefficient and wastes resources while running the MapReduce procedure.

2 Part B - Big Data Project Analysis

2.1 Task One

AgrBIG has asked for a Big Data architecture with data that is expected to be 300 Petabytes in size and that will be coming from various sources and both structured and unstructured data. These attributes make this a perfect candidate project to implement a Big Data solution due to the volume of data, velocity of data, variety of data, by ingesting data and processing it in the correct way, knowledge and value can be added to this data to the users.

2.1.1 Data Sources

Big Data uses multiple sources of data and the AgrBIG problem is no different. For example real time data can be gathered from Internet of Things (IoT) sensor data that monitors soil composition or various product prices taken real time from market data. Historical weather data can be held in a structured database or social media posts containing text, image data from satellites and drone data provide unstructured data.

These data sources are varied and all provide a different velocity of data to be captured and ingested into a Big Data solution and provide the data that will be consumed for the entire solution. Hadoop is an excellent solution for Big Data but it is a write once read many times distributed solution and does not lend itself well to real time data sources, another solution that is suited to real time data ingestion is HBase, which has the option of in memory processing which is many times faster than read/write from a physical disk. Storing data in regions and tables and when tables become to large, distribute these tables into smaller tables with new regions.

As there are multiple sources of data, to get the best performance and solution streaming data could be consumed by HBase leveraging its ability to handle this type of data and Hadoop for batch processing large quantities of data such as historical weather patterns.

2.1.2 Data Extraction and Cleaning

As the data fed in from IoT devices can be done in real time, occasionally there could be missing values or corrupt data which could skew any analysis performed. This is not only for IoT devices, it could also happen from any type of data source, for instance the entire United Kingdom has recently completed a Census for 2021, but not all questions that have been asked are mandatory which will lead to missing values from those who do not wish to answer these questions. Data cleaning is the process of removing data that is incomplete, to prepare data for processing tools can be used to identify these values such as AWS Glue DataBrew. This tool allows the user to visually inspect data that is stored and has over has *"250 pre-built transformations to automate data preparation tasks"* (Harrison, Chapman, and Beaton, 1997) and a claim of faster cleaning and normalisation of data by up to 80%.

2.1.3 Data Storage

Data warehouses can store Petabytes of data, but in a structured format that is pre processed ready to be used in analysis. With a schema on write system, this means that the data has a schema before any data is read from the system. As its is highly structured, it is not ideal for storing streaming data and cannot handle unstructured data that could be coming from images, text or video. With this structure, agility in data is also lost as configuration is completed during the installation and changing this configuration can be costly and time consuming. It does however have an advantage of security which could benefit AgrBIG with any data that might need to be kept more secure.

To store this variation and volume of data, Data Lakes are key to getting most value from data. Data Lakes are schema on read, so data structures are not defined until analysis is required. These stores allows storage of data from IoT devices, social media post, which are considered unstructured data and also data from traditional relational databases also. Raw data is stored within Data Lakes allowing analysis to be performed on this data at a relatively lower cost then a traditional solution. Data Lakes are highly agile and can be configured to suit tasks often and without much time lost when compared to Data Warehouses

2.1.4 Batch Processing

Batch processing is using data that has been collected over a period of time to be used in analytical exploration, this means that is is stored before use and any value of data is extracted. For instance ArgBIG would like to hold historical data on weather patterns which could be processed in a batch to get useful insights from it. Hadoop is very useful for this task of batch processing data and extracting the value from historical data. With a distributed architecture Hadoop can make use of commodity hardware which keeps costs lower.

2.1.5 Real Time Massage Ingestion

Real time injection of data is also important to AgrBIG, the solution for this data ingestion would be to use Apache Kafka to consume this data and split data for real time processing or storage in Data Lakes ready for data cleaning and extraction. This solution is distributed across one or many nodes to leverage parallel processing power and is also open source.

2.1.6 Analytical Data Store

Analytical data stores hold past information that has already been processed and can be used again for comparison information, predictions or planning. These examples are not a complete list of what analytical data stores are used for but just a few examples on the value that can be held within this data and extracted if looked for. Data visualisations can be created from this storage of data.

2.2 Task Two

AgrBig would like to have a data store that hold a large volume of plants, crops, diseases, symptoms and pest. This type of data store would be very well suited to a graph database that stores relationships between data also known as nodes rather then ridged tables. A relational database is too ridged for storing data in the same way the a graph database does and would require large complex joins between tables along with more processing power and time to complete the operation.

Graph databases such as Neo4J store nodes with attributes, which are key value pairs, that make up the properties for that node and labels that allow the representation of roles within the graph. This would allow AgrBIG to build a graph model from the data that they currently have about corn diseases. The relationships that connect nodes are directional in nature, and a single node can have many different relationships with multiple nodes. These relationships also have properties which are relative to the real world relationship which can be used to show the relationship between diseases that are caused directly or indirectly by zinc deficiency's by implementing Cyphers Shortest Path function that computes shortest paths between directly disconnected nodes but share nodes that connect them by relationship.

2.3 Task Three

As Hadoop and MapReduce do not support standard queries written in Structured Query Language (SQL) people who are used to using these queries are at a disadvantage when moving over to a Big Data solution. However there are solutions that are suited to help SQL programmers transition over to MapReduce to perform queries on Big Data. An alternative way for AgrBIG employees who are more accustomed to SQL would be to implement Apache Hive, which has its own programming language HiveQL which is similar to SQL and will help developers migrate faster.

Hive is a data warehousing solution and as stated before has HiveQL as its programming language, the architecture for Hive is built on top of MapReduce and Hadoop and implements a execution engine that can convert HiveQL queries to MapReduce jobs which will fetch the desired data then present the results back to the user. To achieve this the workflow between Hive and Hadoop are as follows:

1. Query sent to driver from interface.
2. Compile the query.
3. Compiler sends metadata request to Metastore.
4. Metastore sends metadata back.
5. Resend plan back to the driver.
6. Driver sends execution plan to execution engine.

7. Execution of job.
8. Results are received from Data nodes.
9. Execution engine sends result values to driver.
10. Driver sends results to interface.

From these steps we can see that there is a MetaStore which is a component that stores metadata on data location and table structure and schema for Hive separately from the Hadoop Distributed File System (HDFS). Although these steps look intimidating,the user only executes the first step of sending the query that is written in HiveQL and the rest is taken care of by the Hive frame work, making it an ideal fit for AgrBig SQL developers.

2.4 Task Four

Real time stream or near real time processing is using the data that is provided at the time it is created, however MapReduce is not designed for continuous iterative computational processing as it will kill any process as soon as it completes it task. This fact is less then ideal for stream processing data, however there is a solution in Apache Spark which is designed to be used on the same type of distributed systems that MapReduce also takes advantage of like HDFS, but uses memory caching to speed up execution by reducing read/write operations to slower physical disks. The Apache Spark ecosystem has a dedicate module called Spark Streaming which can process real time data from sensors that could be connected and either write this data to disk to be used for analysis at a later date or displayed on a dashboard for real time updates and interaction for the end user.

There are two primary types of operations that can be performed on RDDs, one of which are Actions. Actions do not change RDDs, but they perform evaluations such as aggregation or counting specific items of data. The second operation is transformations which crate a new RDD from an existing RDD but with changes to data.

Spark Streaming uses Resilient Distributed Datasets (RDD) as a structure to hold the information that is going to be processed, with failure events this system can rebuild information as it tracks history making it Resilient. With data spread over multiple nodes parallel processing can commodity hardware is made possible which covers Distributed and Datasets covers data that is grouped and ready to be processed. Spark partitions RDDs automatically across nodes so it does not need an employee to figure out how to split data across multiple nodes ready for parallel computations. Which in turn reduces the workload and speeds up the data pipeline for processed data.

2.5 Task Five

Hosting a global, highly available and can scale according to the demand would lean heavily towards using a cloud based service such as Amazon Web Services.

Reason for this being that these services are available world wide, the resources that are needed can be expanded and contracted as demand grows and shrinks which will save on outlays on hardware.

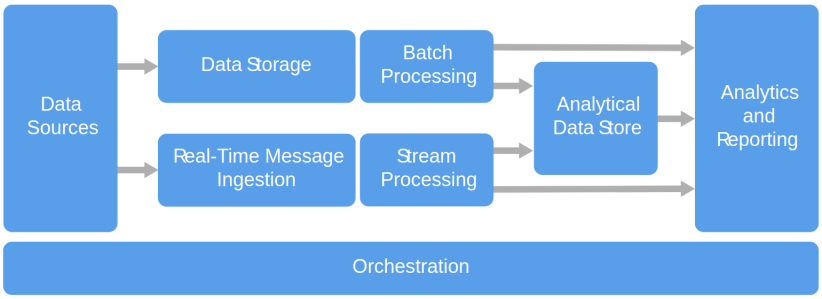

Figure 1: *"Big Data Architecture"*(Microsoft, 2019)

By splitting data sources into two separate pipelines, we can process data in real time and with batch processing. For the batch processing pipeline data incoming from various sources can be stored in a Data Lake , namely the Amazon S3. From this S3 storage node data can be passed to a Amazon Elastic Compute Cloud (EC2) node which can process data using Amazon EMR which allows the use for Hadoop and Spark across node clusters. For data cleaning AWS Glue DataBrew can be implemented, as this has automated preparation and transformations built in for users to implement. With processed, transformed and clean data, data and visualisations can be pushed to a dashboard for end users to analyse to learn knowledge that has been extracted.

For streaming data such as IoT devices, real time ingestion of unstructured data is required. To capture this data Amazon Kinesis can be implemented which has modules for video streaming such as Kinesis Video Streams, data streams in Kinesis Data Streams. Using Kinesis Data FireHose data can be processes and stored in a Data Lake for use at a later date if real time data is not required for that particular segment of data. For data that AgrBIG would like real time information, Kinesis Data Analytics provides a platform for real time reports.

To store processed data from both batch data and streaming data, Amazon DynamoDB can be used as a analytical store, this particular database is a NoSQL database that can have both key value pairs and document based databases and can be scaled to AgrBIG current need to store data. This analytical data store can be queried at later dates to help show, but not limited to, predictions and performance of the business.

Reference

Harrison, Paul, Nicky Chapman, and Clare Beaton (1997). *Glue*. URL: https://aws.amazon.com/glue/features/databrew/.

Microsoft (Nov. 2019). *big data architecture*. Microsoft. URL: https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/big-data.

Bibliography

Bakshi, K. (2012). "Considerations for big data: Architecture and approach". In: *2012 IEEE Aerospace Conference*. Accessed 01-03-2021, pp. 1–7. DOI: 10.1109/AERO.2012.6187357.