Miriam Garcia

ADTA 5240:

Harvesting, Storing, and Retrieving Data – Midterm Assessment

**PART I: Apache Hadoop Framework in Cloud**

Apache Hadoop is an open source framework that is capable of processing vast amounts of heterogeneous data sets across clusters of commodity computers and hardware. Hadoop uses a simplified programming model. One of its major advantage is that it provides a reliable shared storage and analysis system, in this case in the Google Cloud Platform.

Google Cloud Platform, or GCP, is a public cloud service provider. It is an Infrastructure-as-a-service (IaaS) that provides hosts and maintains core infrastructure. Using Apache Hadoop in a cloud provides multiple advantages such as agility, flexibility, efficiency, savings, and more.

A screenshot of a cell phone screen with text

Description automatically generated

In Google Cloud Platform we can look at our VM instances that are running. There are 3 nodes associated within this cluster, a master node and two worker nodes in this case.

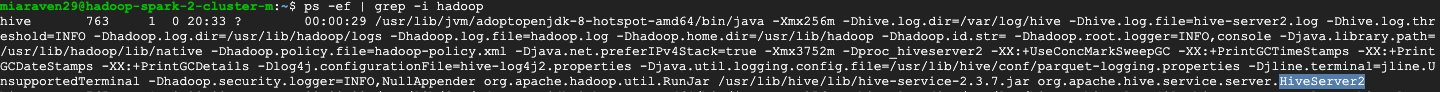
The Apache Hadoop Ecosystem is a suite that is constructed of various tools that bring forwards solutions to big data problems. There are major components within the Apache Hadoop Ecosystem, some of the most vital being: Hadoop Distributed File System (HDFS), MapReduce, YARN, Hive, and Spark. The Hadoop Ecosystem provides reliable shared storage through HDFS and a strong analysis system via MapReduce and Spark.

Hadoop/Spark subsystem name: hive

Process ID: 763

Name of subsystem running: HiveServer2

The Hive subsystem makes reading, writing, and managing large datasets accessible by residing in distributed storage. SQL language can then be used to perform distinct tasks on the evaluation and access of this data. A JDBC driver and command line are used in order to connect users to the Hive.

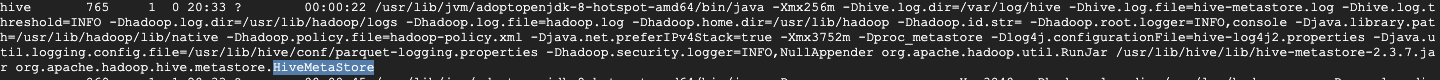
****

Hadoop/Spark subsystem name: hive

Process ID: 765

Name of subsystem running: HiveMetaStore

The Hive MetaStore is the central repository of Apache Hive metadata. Its function is to store metadata for Hive tables and partitions in the relational database. It also provides easy client access to this information such as the tables schema by using an API.



A screenshot of a cell phone

Description automatically generated

In this graphic we can see how the Hive architecture is structured. Using a JDBC driver we can request access to the HiveServer(2), which then allows us to do lookup or update the metastore server.

Hadoop/Spark subsystem name: spark

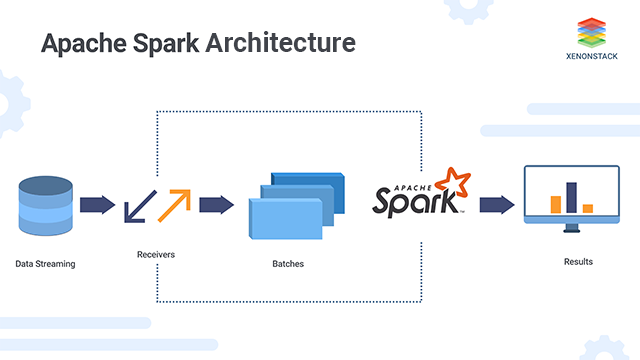
Process id: 1854

Name of subsystem running: HistoryServer

Spark distributes data across a cluster then processes said data in parallel. It also shuffles files around on disk and since Spark runs on memory it is faster at processing data. The Spark subsystem can run on Hadoop clusters, its own independent cluster, or in cloud based platforms such as GCP. Spark can also access a large variety of data sources: HDFS, Apache Cassandra, Apache HBase, and Cloud-based storage. We can also run a cloud-native managed Spark as a subsystem that helps us reduce errors and provides flexibility. A Spark cluster can be launched with a GUI tool in the Google cloud console.

**A screenshot of a cell phone

Description automatically generated**



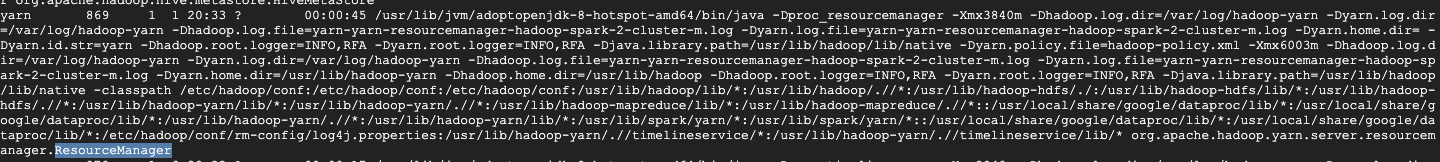
This graphic simplifies how the Apache Spark Architecture is structured. First we begin with streaming data, the receivers then send the commands towards the batch processing within Apache Spark, which afterwards gives us the results. In short, Spark is an analytics engine for large-scale data processing with various tools within.

The following show the subsystems running on GCP and will be expanded more on PART III.

Hadoop/Spark subsystem name: yarn

Process id: 869

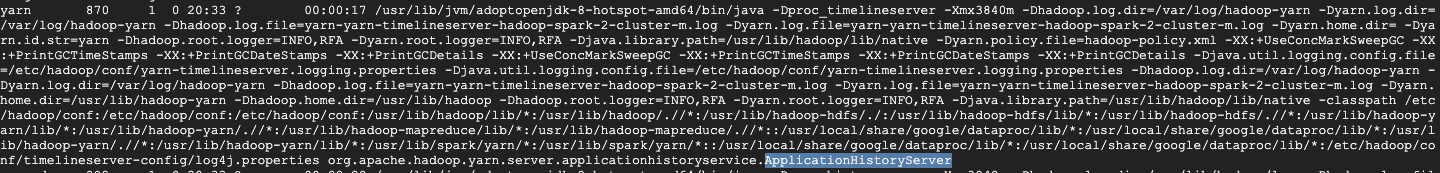
Name of subsystem running: Resourcemanager

****

Hadoop/Spark subsystem name: yarn

Process id: 870

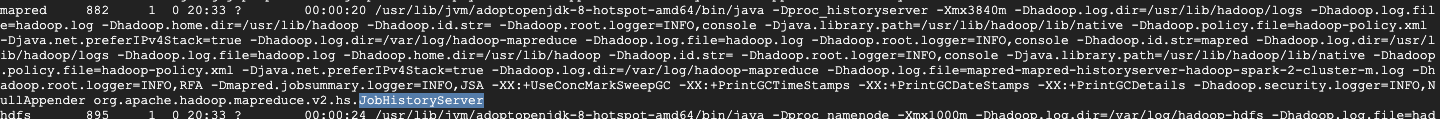
Name of subsystem running: ApplicationHistoryServer



Hadoop/Spark subsystem name: mapred

Process id: 882

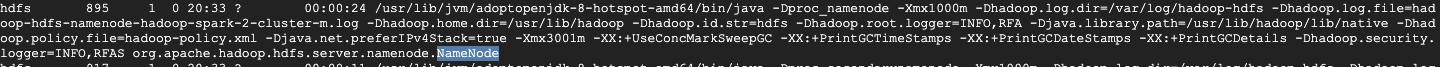
Name of subsystem running: JobHistoryServer



Hadoop/Spark subsystem name: hdfs

Process id: 895

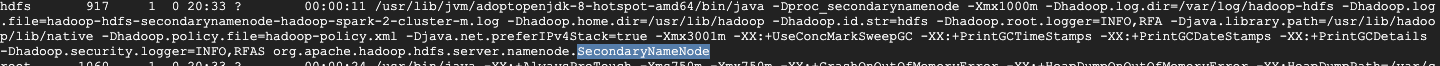
Name of subsystem running: Namenode



Hadoop/Spark subsystem name: hdfs

Process id: 917

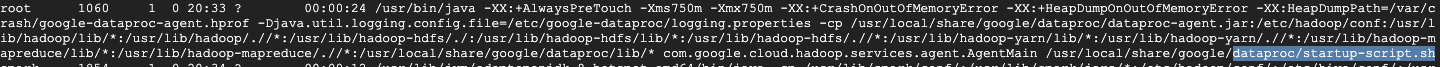
Name of subsystem running: SecondaryNameNode



Hadoop/Spark subsystem name: root

Process id: 1060

Name of subsystem running: dataproc/startup-script.sh



## PART II: Fundamental Concepts of Data Engineering

### Structured, Unstructured, and Semi-structured Data

### Structured Data

### Structured data is standardized data that can be easily categorized and organized. It is usually text files that is displayed in titled rows and columns. Said structured data can be easily processed and ordered with data mining tools. This type of data makes it easy to search within it in relational databases because it has a defined fixed schema. Structured data can be related to other data records easily within its structure as it is mostly predetermined records and fixed fields. The facilities that structured data provide mostly relate on how easy it is to upload information once it has been organized into a relational database. It is easy to search within it and from there to enter commands. It is possible to enter, store, query, and analyze structured data without having to worry about formatting it or getting values that have no “text” or are un-readable. That being said this structured data must be defined in terms of a field name and what type it is such as alpha, numeric, date, currency, etc.

### Unstructured Data

### Unstructured Data on the other hand is information that has no predefined data models or has not been organized in a predefined manner or a fixed schema. It can be textual or non-textual and human or machine generated. Unstructured data differs greatly from structured data, noting that it has no schema and no transaction management. It also differs in that only textual queries are possible and instead of relational databases, characters and binary data are in place. The issue with unstructured data is that it results in irregularities within that make it complicated to understand and analyze the data using traditional programs. Examples of unstructured data include but are not limited to:

### Text files (word processing, presentations, emails)

### Social Media content (data from LinkedIn, Twitter, Facebook)

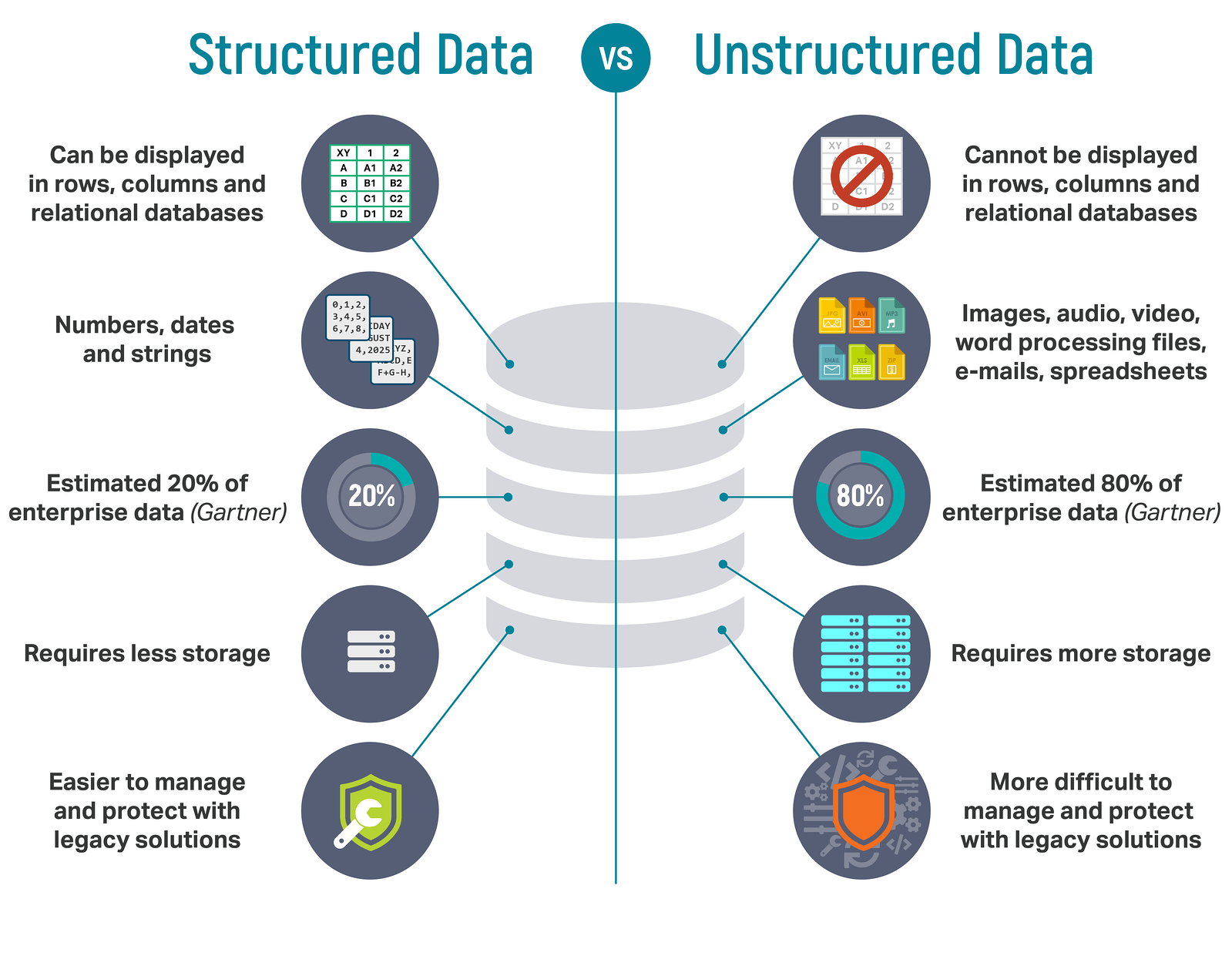
### Websites (YouTube, Reddit)

### Media (audio and video files, digital photos)

### Satellite imagery (weather data)

### What's Hiding in Your Unstructured Data? | ORI

From this graphic, we can see that a lot can be hiding within unstructured data. This data can be vital for the firm’s evaluations of its performance, ROIs, and more.



This graphic sums up some of the main differences between structured and unstructured data. In todays’ world, a significant percentage of data is unstructured. This makes it vital for us to use and develop more tools to analyze unstructured data correctly.

### Semi-Structured Data

### Semi-structured data is information that doesn’t reside inside a relational database but still maintains some organizational properties that facilitate its analysis. Semi-structured data is not as rigidly defined as structured data. Most importantly it maintains internal tags and markings that can be used to identify data elements and enable hierarchies and groupings. It is not as unmanageable as unstructured data. Examples include:

### Emails (as a whole)

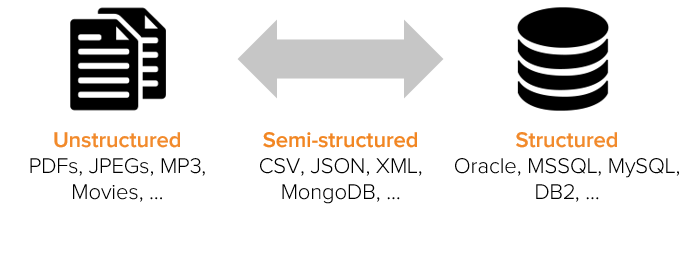
### Comma Separated Values (CSV files)

### XML format

### JSON formatted data

### NoSQL databases data

### Tab delimited files



This graphic summarizes where different sources of data go into their respective category.

### Concepts of SQL and relational databases

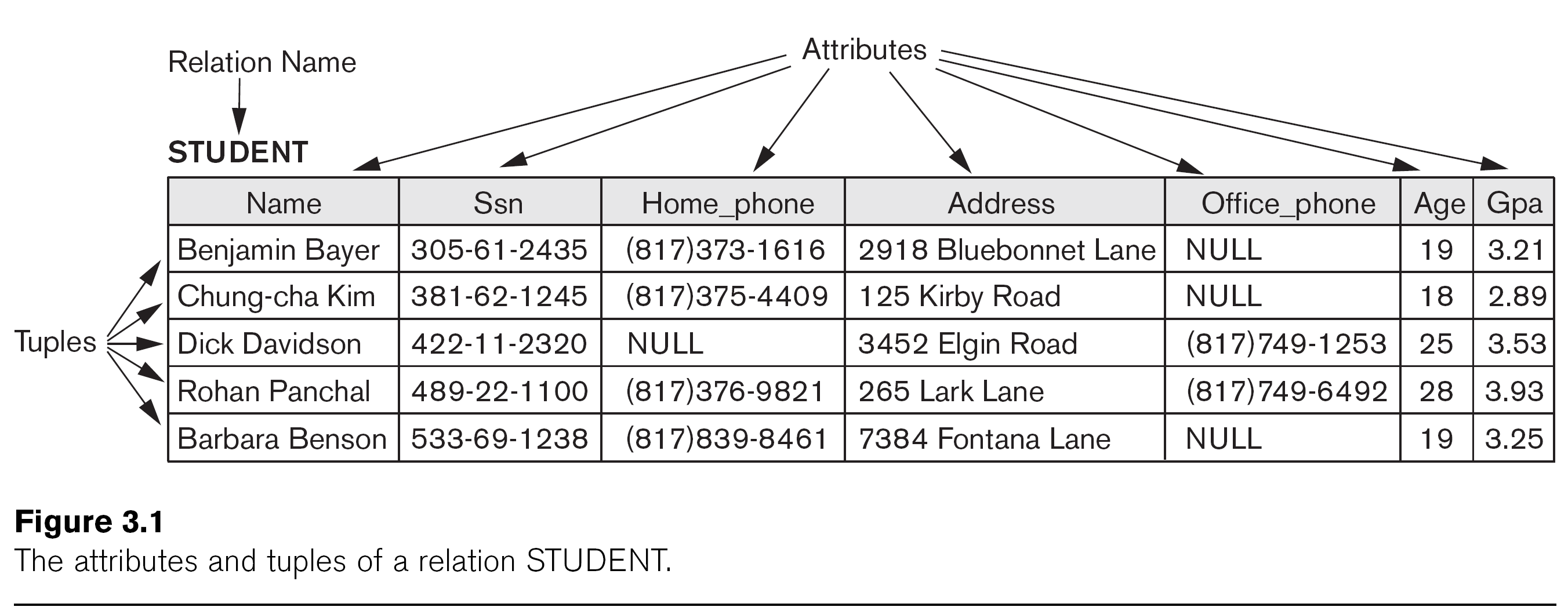
Structured Query Language, or best known as SQL, is a programming language that was designed for managing data in relational database management systems (RDMS). The SQL language can be used to write code in programs that can manage and create databases. A relational database is a type of database that uses a structure that helps us access data and connect it in relation to another piece of data within the database. The structure in RDMS makes it easy to find the connection in between the data since it is organized into distinct tables with connections. A Relational Database Management System, or simply known as RDBMS is a program that allows you to administer, update, and create a relational database. The connection between RDMS and SQL is that most RDMS use the SQL language in order to access the Diagram

Description automatically generateddatabase.

This graphic elaborates on how once we input an SQL query statement, it will be sent to the RDBMS server which will then give us the requested output.

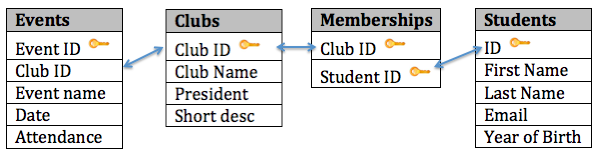
A relational database organizes data into linked tables that are related based on data in common. The main advantage of relational databases is how they allow users to easily categorize and store data to query, filter, and extract information from. These databases are not reliant on physical organization and a new data category can be later added. Some other advantages of a relational database include the elimination of data deduplication, flexibility for complex queries, the access of the same database by multiple users, and the security aspect of how data in tables within it can be limited to access to specific users.

A relational data model represents data as a table where each row (tuple) represents a single record. Each column represents a single variable/field. In a relational data model, a database is simply a collection of tables (relations).



Here we can see the relation name, the rows (tuples), the attributes with their name and data type, and the table’s overall schema.

As we know, a relational database stores data in tables. These tables are organized into columns, and each column stores one type of data whether it be an integer, a real number, character strings, date, or more. The best way to manipulate a relational database is to submit SQL statements. It must be noted that relational databases are designed for data integrity. The data for a single “instance” of a table is stored as a row, which means that duplicates are not allowed in terms of rows. Those tables usually have keys that allow us to uniquely identify a row within a table. There are primary and foreign keys in most cases when we expand to looking at more tables. Where the primary key is one or more fields that uniquely identifies a row in a table and a foreign key is a relationship between columns in two database tables.



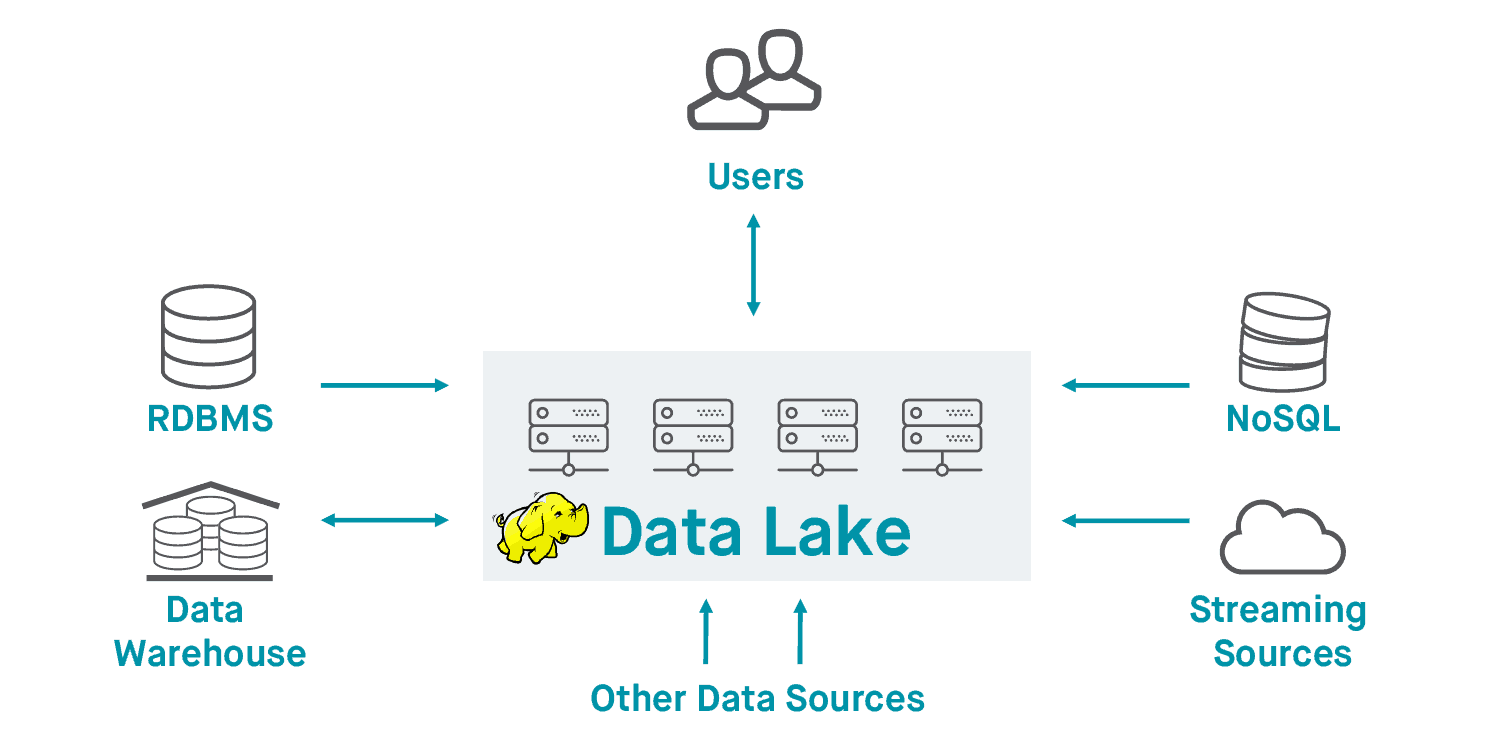
Here we can see how each table has its unique primary key such as Event ID in the Events table. We can also see the foreign key Club ID within the Memberships table.Diagram

Description automatically generated

Here we have an alternative table that elaborates on the characteristics of a relational table, where we can easily identify the primary key of this table.

**Data Lakes**

A data lake is a centralized storage repository that allows you to store structured, unstructured, and semi-structured data at any scale. That is, data lakes allow us to store raw data in its native format. Data Lakes are becoming more common as their economical cost allow firms to explore their data through these repositories. Firms are now using data lakes to complement their data warehouses. The idea of implementation of data lakes as storage repositories involves the maximization of Return On Investment (ROIs) as firms can derive insights from both structured and unstructured data.

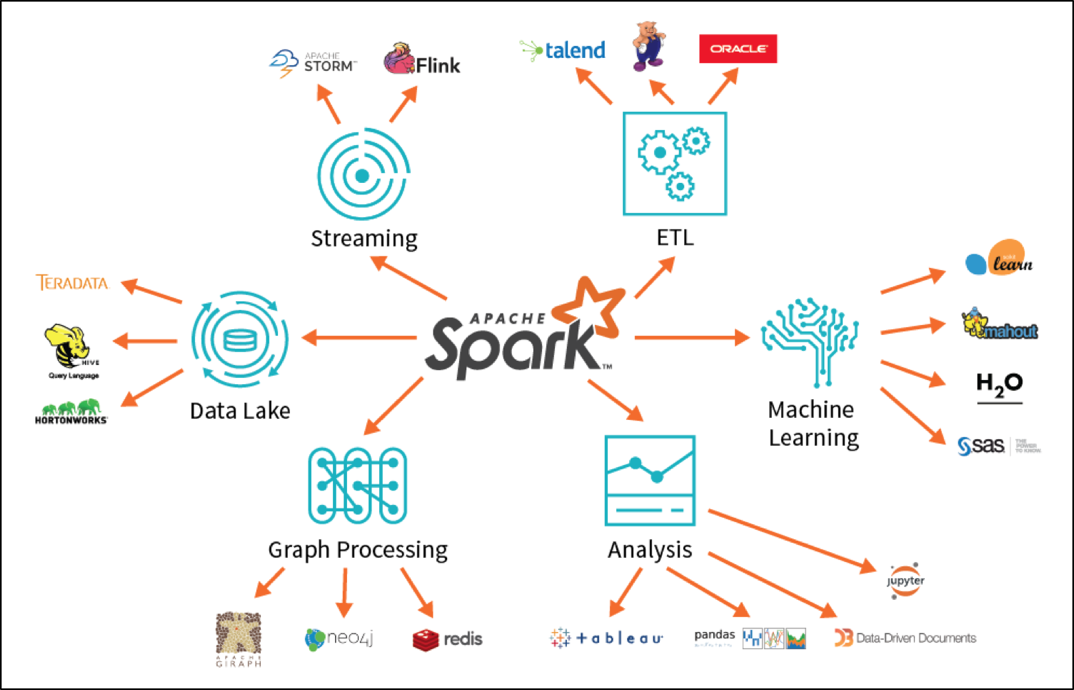
The concept of a data lake comes from what an actual lake would be, where the incoming flow of information comes from a vast amount of raw data. Then from there the “water” within the lake is the reservoir of data that is stored in order to perform analysis. The outflow of the data then is after it has been analyzed. In the Apache Hadoop ecosystem, data lakes can be built within its open sourced projects. Data Lakes allow you to store relational data just like operational databases would. They also allow you to import data in real-time. From there it is easy to scale data without wasting time on defining data structures, transformations, and schemas. In Apache Hadoop, a data lake architecture involves storing data in the HDFS across a set of clustered nodes.

This graphic helps us look at how relationships are established and how information flows within the Hadoop Data Lake architecture (Thomas, 2017).

Timeline

Description automatically generated

This graphic simplifies the process of using the SQL language on Hadoop Architecture and Process Flow. After sending an SQL query, through the Data Lake our queries are processed on each node, results are collated, then we get the query results in return (Thomas, 2017).



This graphic summarizes the various functions of Apache Spark and its components. We can use it through Data Lakes, we can use it for Analysis and Machine Learning.

**Differences between Data Lakes and Data Warehouses**

Even though the function of both data lakes and data warehouses is to store big data there are distinct key differences. A data lake might work best for a firm, while a data warehouse might work for another, or for other firms a combination of both helps them optimize data storage and analysis. The biggest distinction between a data lake and a data warehouse is that data lakes can have all types of primary data—structured, raw unstructured, and semi-structured. In comparison, a data warehouse can only take structured data. In terms of scalability, data lakes can hold any amount of data at a low cost regardless of the data’s type. While scaling in a data warehouse becomes exponentially more expensive. The schema of a data lake is actually written at the time of analysis, better known as schema-on-read. Versus the schema for a data warehouse is a schema-on-write, which means that it is designed prior to the data warehouse implementation.

**Data Processing methods**

Loading data into a data management system can be either schema-on-write or schema-on-read.

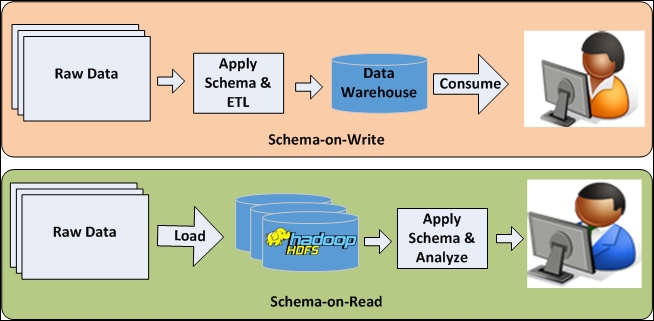
**Schema-on-write**

Schema-on-write is the process that involves creating a schema for the data before writing it into the database. In other words, the schema must be written way before uploading our data into a database. When we first begin working with a relational database, the first step is to create tables and configure the schema. This approach involves defining columns, the relationships of columns, and data format before the data is uploaded. One of the biggest limitations is that we cannot upload any data into the database until the appropriate schema of the data has been formatted and written. To do this we need to understand the entities of this data in order to reflect their relationships in the tables.

You must not only define the schema for the data, you also have to structure it based on that schema. In order to do any development in the database we have to use SQL to read data from the database, which ties back into the Relational Database management system. With the RDMS, we have to perform what is known as an ETL Pipeline—which stands for Extract Transform Load. An ETL Pipeline refers to the processes involving the extraction of data from the input source, the transformation of the data, and the loading into the output destination (the database). With schema-on-write, the most important step in the ETL pipeline is the transformation of the input data into a form that the output destination can store.

**Schema-on-read**

Schema-on-read follows fast data ingestion since the data does not follow any internal schema. It is easier for us to upload the data since it is essentially just copying and moving files into the database. It isn’t necessary to define the schema before storing it, which makes it easier to upload both structured and unstructured data. The actual schema is created only when the data is actually being read. Schema-on-read is much more flexible than schema-on-write since it allows us to perform frequent schema changes. Schema-on-read usually follows the path of uploading data as it is regardless of structure and then applying your own schema to the data in order to read it back out. The Apache Hadoop ecosystem is actually schema-on-read since MapReduce jobs make sense of the data as it is being read. The schema is inferred on the go as data is being read off the file system for analysis. Since the majority of data nowadays is unstructured data, it is important to know the benefits of shifting to the schema-on-read process.



A simplified graphic that shows us the difference between schema-on-write and schema-on-read. Schema-on-write involves the application of a schema and ETL Pipeline that then takes it to the data warehouse. Versus for schema-on-read we have to load the raw data into Hadoop HDFS, then we apply the schema and analyze.

## PART III: Apache Hadoop Ecosystem

Apache Hadoop is an open source framework that aids in the processing of large data sets which reside in the form of clusters. As a vital framework, Hadoop consists of several modules that are supported within its large ecosystem. The Apache Hadoop Ecosystem is comprised of different components such as HDFS, YARN, MapReduce, Spark, Hive, PIG, HBase, Zookeeper, and more.

**HDFS (Hadoop Distributed File System)**

HDFS is the primary component of the Apache Hadoop ecosystem. It is responsible for the storage of large data sets (both structured and unstructured data) across various nodes. This component maintains the metadata as log files. It creates several replicas of the data block to be distributed across different clusters in order to access data promptly. HDFS is the component that maintains all of the coordination between clusters and hardware and hence acts as the “heart” of the system.

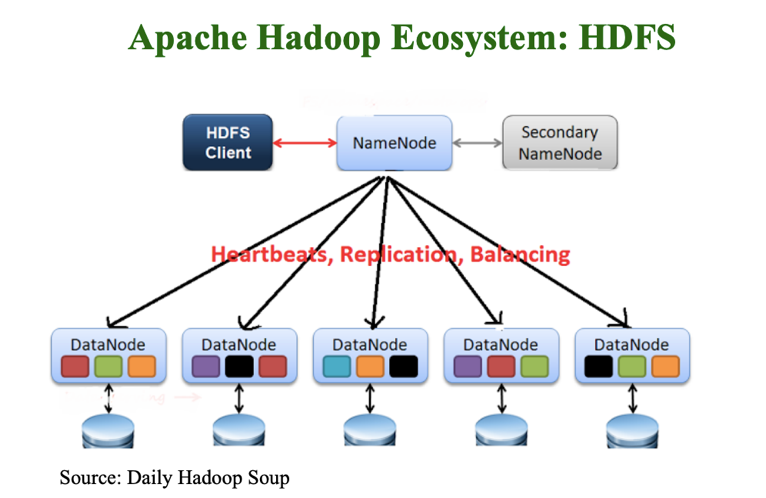
HDFS consists of three core components:

•Name Node

•Data Node

•Secondary NameNode

It is also important to note that HDFS operates on a Master-Slave architecture Model where Name Node acts as the master node that keeps track of the storage cluster and the Data Node acts as the slave node, which sums up to the various systems within the Apache Hadoop cluster. The Secondary Name Node works with the primary Name Node as a helper daemon and performs checkpointing.



With this graphic, we can see how Name Node is the primary node that contains the metadata. These data nodes are commodity hardware in the distributed environment.

**YARN (Yet Another Resource Negotiator)**

YARN helps manage the resources across clusters. It performs resource allocation and scheduling within the Apache Hadoop System. Compared to HDFS which splits up the data storage across the cluster, YARN splits up the computation functionality. YARN is responsible for aligning worker and storage nodes in order to run jobs efficiently.

YARN consists of three major components:

* Resource Manager
* Application Master/Manager
* Nodes Manager

The Resource Manager is the ultimate authority that allocates resources for the applications in the system. The Node Manager is the per-machine framework agent that is responsible for containers, monitoring their resource usage, and also reporting back to the same Resource Manager. The Application Master is a framework specific library that negates with the resources from the Resource Manager and works with the Node Managers in order to execute and monitor tasks (Apache Hadoop YARN, 2018).

A screenshot of a cell phone

Description automatically generated

**A screenshot of a social media post

Description automatically generated**In this graphic we can see that the Resource Manager has two main components, which are Scheduler and Applications Manager. The Node Managers provide Node Status to the Resource Manager. We can also note that the Application Masters have the responsibility of negotiating appropriate resource requests, tracking their status. and monitoring for progress.

**A screenshot of a cell phone

Description automatically generated**

In GCP, we can easily keep track of the YARN memory, YARN pending memory, YARN Node Managers and HDFS capacity.

**Yarn in Hadoop 2.0**

Hadoop 2.0 was introduced in 2013 with YARN as its most vital added component. Hadoop YARN is a significant upgrade from Hadoop 1.0 since it provides performance enhancements that benefit all other technologies connected in the Hadoop Ecosystem. It interacts with the Hadoop database and the Hive data warehouse. In Hadoop 2, the resource negotiation part was split out from MapReduce, so that alternatives to MapReduce like Spark could be built on top of YARN.

In the previous version Hadoop 1.0 (MapReduce Version 1), MapReduce used to perform both the task of process and resource management by itself. It had a job tracker module that worked as a single master that allocated resources for applications and performed scheduling along with monitoring jobs of processing in the system. The problem with this system was that the design of a single master for all components resulted in a bottlenecking issue and inefficient computational resource utilization. This is where YARN comes into place. The main reason for this technology was to help separate MapReduce from Resource Management and Job scheduling instead of having a single master. This is the reason why YARN is responsible for Job scheduling and Resource Management (Khanvilkar, S).

A screenshot of a cell phone

Description automatically generated

Graphic shows comparison of how YARN is intricately woven into Hadoop 2.0.

**MapReduce**

MapReduce is Hadoop’s original framework for writing parallel applications that process vast amounts of data—both structured and unstructured—in HDFS. The way it can perform said monumental tasks is by utilizing the location of the data and processing it near where the data is stored on each node cluster, which in turn reduces the distance which it must be transmitted to. Compared to traditional storage systems that use centralized servers that create bottleneck issues, MapReduce aids in the “flow” of the data.

The way MapReduce functions is that it breaks down monumental tasks into smaller tasks, then assigns them to different systems. When all tasks are processed, the output each computer transitions into a single location and then an output dataset is arranged and solved. MapReduce uses parallel and distributed algorithms that transmit the processing logic. This allows us to write applications that break down big data sets.

MapReduce has two main functions:

* **Map()** which performs sorting and filtering of data, which organizes data by grouping. The Map function transforms input data into key and value pairs. Map generates a key-value pair based result that is then processed by the Reduce() method. Maps here are the individual tasks that transform input records into intermediate records.
* A close up of text on a white background

  Description automatically generated**Reduce(),** is the functionality that then aggregates the mapped data. Reduce reduces a set of intermediate values that share a key into a smaller set of values. Reduce takes the output generated by Map and reduces functions to each group of values that share the same key and then writes the reduce output back to HDFS.

In this graphic we can see how MapReduce breaks down data loads through first Map(), sorting Locally, combining, map writing back into the system. Then reduces tasks by merging and combining, Reduce(), final writing, and shuffling (Li, Mazur, et al).

**PART IV: HDFS – MapReduce – HIVE: Simple Schema & Query**

A screenshot of a cell phone

Description automatically generatedA simple schema was created for the data file userdata.csv. The below commands were executed on the command window of the master node and lists the files which were imported from the GCP. A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

We then connect to the Apache Hive by running the beeline command

beeline -u jdbc:hive2://localhost:10000

The below command provides us the commands:

* To create external table
* Display the contents of the first row of the data file
* Display the contents of the first 3 row of the data file

A screenshot of text

Description automatically generated

Here we have a screenshot that provides the execution of various commands. And now four folders are within the system:

* Final
* Test
* userdata
* weblog

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

**PART V: HDFS – MapReduce – HIVE: Detailed Schema & Query**

We then create a schema, or an external table, for the data file userdata.csv saved in HDFS. This schema is now well-defined with all the variables of the data set according to the respective source file.

Columns are created based on the schema of the file as:

* Date
* Userid
* Firstname
* Lastname
* Location

New table userdata\_top5 table has been created.

A screenshot of a cell phone

Description automatically generated

Running a query in order to display the top 5 locations (top: appearing the most) that can be found in all the records of the collected data set, then ordered by descending alphabetical order.

A screenshot of a cell phone

Description automatically generated

**References**

Apache Hadoop YARN. (2018, November 13). Retrieved September 20, 2020, from <https://hadoop.apache.org/docs/r2.9.2/hadoop-yarn/hadoop-yarn-site/YARN.html>

Chiang, C. (2018, July 31). Structured Data vs Unstructured Data. Retrieved September 25, 2020, from https://www.igneous.io/blog/structured-data-vs-unstructured-data

Khanvilkar, S. (2020, July 17). What Is Hadoop Yarn Architecture & It's Components. Retrieved September 20, 2020, from https://www.upgrad.com/blog/what-is-hadoop-yarn-architecture-its-components/

Li, B., Mazur, E., & Diao, Y. (n.d.). A Platform for Scalable One-Pass Analytics using MapReduce. Retrieved from https://people.cs.umass.edu/~mcgregor/papers/11-sigmod.pdf

MapReduce Tutorial. (2019, August 22). Retrieved September 21, 2020, from https://hadoop.apache.org/docs/r1.2.1/mapred\_tutorial.html

Thomas, S. (2017). Chapter 1: A Brief Overview of the Big Data Ecosystem (Hadoop, Spark, and Beyond). In 947858419 739596561 S. Wooledge (Author), Modern Business Intelligence: Leading the Way to Big Data Success (pp. 1-88). San Mateo, CA: Arcadia Data.