INTERNATIONAL JOURNAL FOR INNOVATIVE RESEARCH IN MULTIDISCIPLINARY FIELD ISSN: 2455-0620 Volume - 6, Issue - 9, Sept – 2020
Monthly, Peer-Reviewed, Refereed, Indexed Journal with IC Value: 86.87 Impact Factor: 6.719
Received Date: 26/08/2020 Acceptance Date: 10/09/2020 Publication Date: 30/09/2020

A BIG DATA PLATFORM ARCHITECTURE FOR ENTERPRISES

[1]Madhusudhan Reddy Sureddy, [2]Prathyusha Yallamula
[1]Associate Director, [2]Vice President
[1] Application Development, Santander Bank
[2] Risk Treasury, Natixis CIB Americas
Email - [1]sureddy21@gmail.com, [2]prathyusha13y@gmail.com

Abstract: *Big data as the name suggests is generally referred to as huge, overwhelming and uncontrollable amounts of information. According to a market study, the global data sphere in 2018 reached 18 zettabytes and the world's data is predicted to become 175 zettabytes by the year 2025. Big data platforms utilizing Apache Hadoop were born to solve the problem of performing analytics on this huge data. Many enterprises quickly caught up to this trend and in their quest of building analytical data lakes, were only successful in building disorganized data swamps due to bad architecture. Big data platforms whether on-premise or cloud are very complex and have many number of tools present in its eco system for handling parallel processing. Huge amount of native coding is also needed to stand them up. In this paper author provides an architecture which simplifies the complexities of the big data platforms and helps enterprises in building a streamlined and clean platform, which is also suitable even for traditional data warehousing needs.*

Key Words: *Big data, Architecture, Enterprises, Platform, Hadoop, Spark, Python, mapreduce, Hive, data layers, ingestion, preparation, analytics, structured, curated, metadata, scheduling*

1. INTRODUCTION:

Big data as the name suggests is generally referred to as huge, overwhelming and uncontrollable amounts of information. Big data is a relative term and the definition of "amount of data" considered as big has continued to change over the years. One of the early usages of "overwhelming amounts of information" was in 1663 when John Graunt was studying bubonic plague which was destructing Europe and he considered the amount of data he had to analyze as huge. Similar example of huge amount of data information was in 1880 when U.S Census bureau was struggling with handling and processing the data collected during 1880 census. Each such usage of the term big data met with introduction of new technologies and ways of handling the then considered big data and also in turn bringing the "amount of data" considered as big higher. In 1663 Graunt introduced statistics and in 1880 Herman Hollerith created Hollerith Tabulating Machine utilizing punch cards design, to solve their respective problems and in turn reduced the time of processing the information by multi fold; Hollerith's machine reduced the census data processing time from 10 years to 3 months. Data volumes continued to grow from 1960s with the introduction of direct-access storage, databases, and online transactional processing systems. Two major inventions and one critical decision changed the landscape of big data definition completely – First invention was when micro computers introduced personal computers in 1977, second invention was in 1989 when British Computer Scientist named Tim Berners-Lee came up with the concept of the World Wide Web (internet) and the critical decision was making internet available for free around the world and only charging for an internet connection by providers. 1990s saw the increased use of internet and 1999 saw the coining of the word internet of things (IOT) and in 2013 IOT comprises of multiple technologies, using the Internet, wireless communications, micro-electromechanical systems (MEMS), and embedded systems. In 2000, the conclusion of an attempt to quantify the digital information by Peter Lyman and Hal Varian was that each year 1.5 billion Giga bytes of data would be produced. This number changed exponentially by the year 2005 with the introduction of web 2.0 - "the user-generated web where the majority of content will be provided by users of services, rather than the service providers themselves". Driver for web 2.0 was the introduction of face book in 2004. According to market intelligence company IDC, the global data sphere in 2018 reached 18 zettabytes and it predicts the world's data will grow to 175 zettabytes by the year 2025.

Exponential growth of data in 2005 also introduced the technology Apache Hadoop which provided the means to process this huge information. Apache Hadoop was introduced by the open source community the apache software foundation and is a collection of open-source software utilities that facilitate using a network of many computers to solve problems involving massive amounts of data and computation. Apache Hadoop was inspired by the Google papers on Map reduce and Google file systems. Apache Hadoop is an eco system consisting of four base modules – Hadoop common, Hadoop distributed file system (HDFS), Hadoop Yarn, Hadoop Mapreduce and multiple additional software packages – Apache Pig, Apache Hive, Apache HBase, Apache Phoenix, Apache Spark, Apache Flink, Apache Kafka, Apache Impala, Apache Flume, Apache Sqoop, Apache Oozie, Apache Storm, Apache Zookeper, Apache Solr, Apache Ambari, Apache Mahout. Cloudera (2008) and Horton works (2011) were the initial commercial distribution providers

that targeted enterprise-class deployments of Hadoop technology on on-premise servers. Cloudera with its merger of Horton works in 2018 continues to be the leading distributor for on-premise Hadoop technology even today. Big data processing through Hadoop also fuelled the rise of cloud computing platforms, as one of the core components of Hadoop was the availability of multiple computers and storage drives at its disposal. Setting up of on-premise servers in organizations was a tedious task which was reduced to mere clicks in the cloud computing platforms. Amazon flagship product Amazon web services became a massive success in 2010s allowing enterprises moving their infrastructure to the cloud platforms. Currently multiple cloud computing providers Amazon web services, Google cloud, Alibaba, Microsoft Azure, Cloudera data Platform and so on are available in the market. Initial big data platform designs with cloud computing utilized the open source Hadoop eco system or Cloudera, Horton works distributions on the cloud servers. More recently the cloud computing providers have also started releasing their own version of big data processing tools on cloud.

Enterprises planning to venture into big data should first understand the three main characteristics of big data and do an exercise on whether their enterprise data falls into those definitions and then only make the decision of moving to big data. Big data characteristics include Volume, Variety, and Velocity. Volume would refer to huge amounts of data – Terabytes or Petabytes that exceed the handling capacity of conventional software. Variety includes the various unconventional types of data – semi-structured and unstructured which are generally not handled by conventional software. Web pages, social networks, internet of things are few of such examples. Velocity refers to the speed at which data is received, processed and decisions need to be made. Traditional systems cannot analyze the huge volumes of data immediately.

Big data platforms currently have two types of implementations – "big data with Apache Hadoop open source tools", "big data with cloud tools". In the big data using Apache Hadoop open source tools organizations build a big data platform on premise or cloud using the Apache Hadoop open source provided by the apache software platform or by utilizing Hadoop distribution providers like Cloudera which were built using Apache Hadoop. Cloud providers can be AWS, Alibaba, Microsoft Azure, and Google cloud and so on. In the big data with cloud tools architecture organizations utilize the cloud tools provided by AWS, Azure and Google cloud instead of the Apache Hadoop native tools for their big data implementations; in some cases like Amazon EMR the cloud based tools could provide some of the Apache Hadoop tool sets. In this paper author covers the architecture model for the "big data Apache Hadoop open source tools"; though the core conceptual architecture of both the implementations are same, due to the tool set differences "big data with cloud tools" has been made out of scope.

2. PROBLEM STATEMENT:

Big data platforms whether on-premise or cloud are very complex due to the number of tools present in the big data eco system and the amount of native coding needed to build them. Tool set options and multiple coding programming languages also adds further complexity in terms of the choice making between the various methodologies and the programming languages. Many enterprises in their quest of building analytical data lakes have only been successful in building disorganized data swamps due to not defining clear data boundaries in their architecture. In this paper author provides an architecture which tries to simplify the build of the big data platforms using Hadoop eco system and also helps organizations to use big data platform for their traditional data warehousing needs.

3. BIG DATA PLATFORM ARCHITECTURE:

Big data platform (BDP) architecture proposed in this paper on a conceptual level has 3 main functional capabilities – Data ingestion, Data preparation and Data insights. These functions further store the data in 5 layers – landing, structured, curated, consumer, and analytics. **Data Ingestion** is the process of sourcing data in raw format from multiple sources using different patterns based on client needs – batch, micro-batch, real-time. The data is write-optimized, indexed and catalogued. Support is provided for multiple data types – Structured, semi-structured, Unstructured. Data is stored in landing layer as part of this function. **Data preparation** is the process of transforming data for analytical and further processing needs. Metadata management and DQ rules are applied on the data to make more sense out of the data and make data ready for analytical capability and vend to other systems. Data is stored in two layers called as structured layer and curated layer as part of this function. **Data insights** is the process of realizing business value out of the data stored and prepared in the data lake. Advance Analytics and Business Intelligence deliver business value from the data taking data level security into consideration. Data integration over multiple subject areas and data sets is applied where necessary. The results are vended to users over Business Intelligence tools or batch based extract or interactive request/response or real-time streaming APIs. Data is stored in two layers called as consumer layer and analytics layer as part of this function. Below illustration provides the information of the various layers in big data with subsequent sections providing information of each layer in detail.

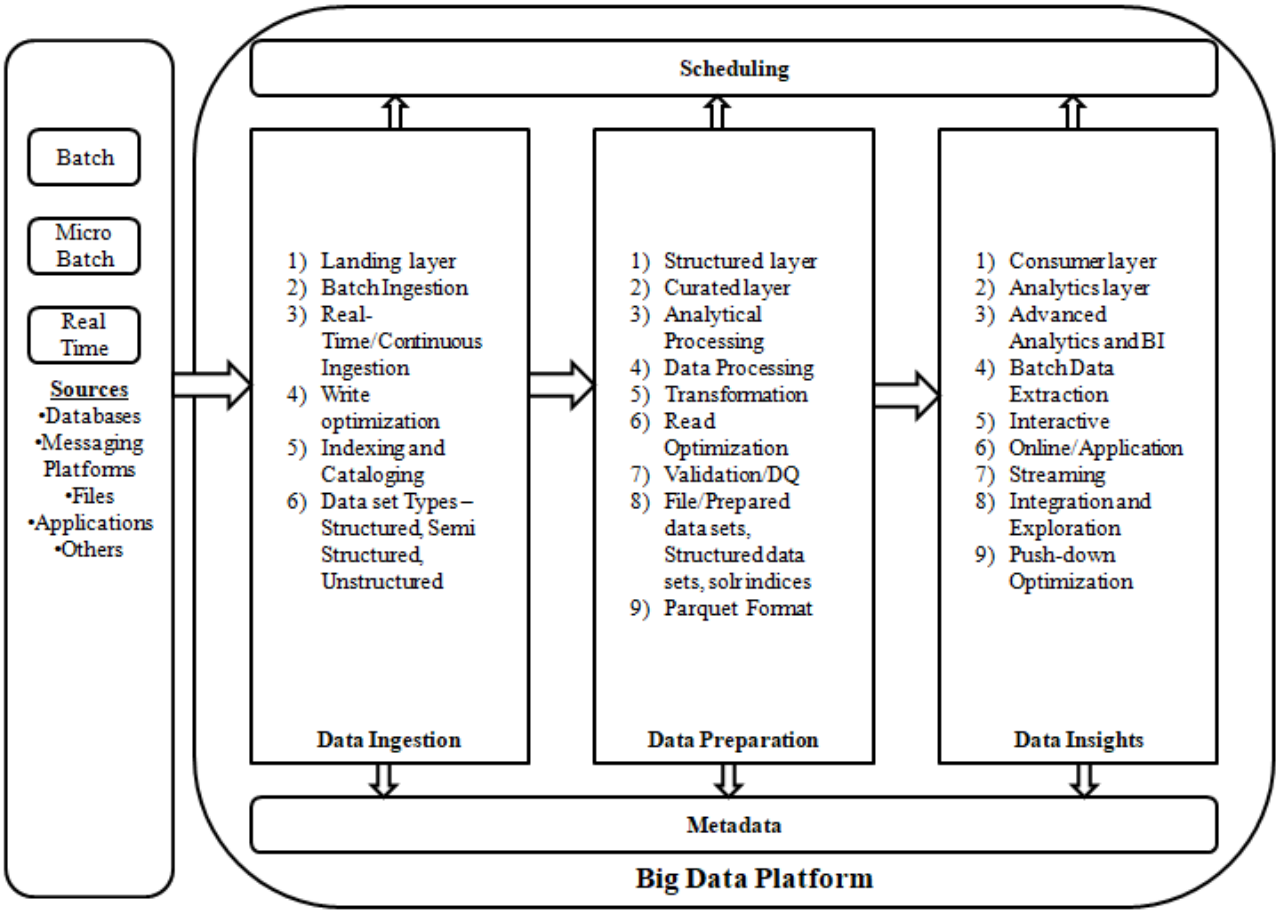

Figure 1: Big Data Platform Architecture

3.1. Data Ingestion: Data which lands into BDP could be of three formats – Structure data, Semi-Structured data, and unstructured data. **Structured Data** is organized into entities that have a defined format, such as XML documents or database tables that conform to a particular predefined schema. This is the realm of RDBMS and lends to schema on write. **Semi-Structured data** is looser and though there may be schema, it is often ignored, so it may be used only as a guide to the structure of the data: for example, a spreadsheet, in which the structure is the grid of cells, although the cells themselves may hold any form of data. **Unstructured data** does not have any particular internal structure: for example, plain text or image. BDP works well on unstructured or semi-structured data because it is designed to interpret the data at processing time, called as schema-on-read. This provides flexibility and avoids costly data loading phase of an RDBMS, since in BDP it is just a file copy. Data landing framework provided in the paper allows multiple patterns to ingest data to BDP. Depending upon the client needs and the sources being dealt with, different patterns can be employed. The patterns can be broadly divided between real-time and batch. Further patterns employ a push or pull mechanisms. In the case of push, sources push data to a message broker or a custom API hosted within BDP. Alternatively in a pull based mechanism, Hadoop native tools are used to pull data from different sources. For instance Sqoop and Flume are employed to pull data from databases and streaming logs respectively. In summary the ingestion patterns can be broken down as below - Real-time data ingestion, Asynchronous data ingestion, direct data pull.

3.2. Real-time data ingestion: This pattern requires organizations to build a custom Application program interface (API) which can respond to real time source system calls. In addition to providing Restful API features, custom API also does multiple tasks as listed below. Tasks vary little between high and low volume data.

Table 1: Tasks for high volume and low volume real time ingestion

Low Volume	High Volume
Real time sources call the custom API. As part of the API, the sources pass document payload and the associated metadata. Source systems are provided with a library for generating supported metadata.	Real time sources calls custom API for sending messages

API performs basic validation on the metadata and payload content. API then generates a unique identifier (UI) for the document based on organization key definitions or UUID. This key will be used for storage in the run-time data catalogue maintained in HBase.	API generates a unique identifier (UI) for the document based on organization key definitions or UUID.
	API publishes messages with the generated document id into Kafka cluster and sends an acknowledgement back to calling real time source with the generated document id. Real time sources listen to the acknowledgement.
Based on use case, payload size and access/vending patterns the data will be stored in HDFS or HBase. Small files which are not used for any structured data vending out of Hive tables will be stored directly in HBase using the UI. In case of HDFS a link to the file will be stored in HBase's runtime catalogue tables.	Based on use case, payload size and access/vending patterns the data will be stored in HDFS or HBase. Small files which are not used for any structured data vending out of Hive tables will be stored directly in HBase using the UI. In case of HDFS a link to the file will be stored in HBase's runtime catalogue tables.
After successful completion of above steps API responds with an acknowledgement with the UI as document id. Real time sources listen to the acknowledgement.	
For supporting search, Hadoop job monitors updates on the HBase table and creates indexes based on metadata and document content	For supporting search, Hadoop job monitors updates on the HBase table and creates indexes based on metadata and document content

3.3. Asynchronous data ingestion:This pattern supports sources that will use file transfer protocols to drop files to a shared file system that is exposed to the custom API. Sources are expected to send payload files and flag files. Guidelines for naming conventions of the payload and the flag files are based on the organization requirements, but have to be defined before setting up BDP. Custom API watches for flag files and the payload file in the shared file system. Custom API based on the details of the flag files will land the files to the appropriate folders in the landing area. Below are the steps to be followed in this pattern:

- Source System send the Raw Documents (payload) and Metadata (flag file) to a shared file system through file transfer protocols
- Custom API will monitor for new files on the shared file system.
- On arrival of a flag file, custom API reads the metadata for the file transmitted from the stored metadata for the file. The metadata contains details like file destination, exception handling etc.
- Custom API checks for metadata content in the flag file. The flag file is expected to have basic content related to the file transmission.
- In case of any exception or missing metadata in flag file, exception handling procedures are followed as per metadata content.
- Custom API lands the file to HDFS landing zone on Hadoop Cluster. Tight security is applied to the landing area.
- Custom API adds a catalog entry in the HBASE document catalog indicating a file has been dropped in HDFS. While doing so, the process generates a document id and uses the document id as the unique identifier (UI)
- Acknowledgement is sent back to the client by publishing the document id back to the source system. Details of acknowledgement is also specified in the metadata content

3.4. Direct data pull – This pattern leverages native Hadoop tools like Sqoop and Flume to source data from client systems directly. Below are the steps which need to be followed:

- Pull data from data bases using Sqoop or from log files using flume. Sqoop and Flume have a capability to integrate with authentication and password storage systems
- Drop data to landing area with appropriate metadata.
- Store metadata in HBase with unique identifier (UI). Small files which are not used for any structured data vending out of Hive tables will be stored directly in HBase using the unique identifier. In case of HDFS, a link to the file will be stored in HBase's runtime catalog tables

- For supporting search, Hadoop job monitors updates on the HBase table and creates indexes based on metadata and document content

Below illustration provides the various patterns involved in the data ingestion:

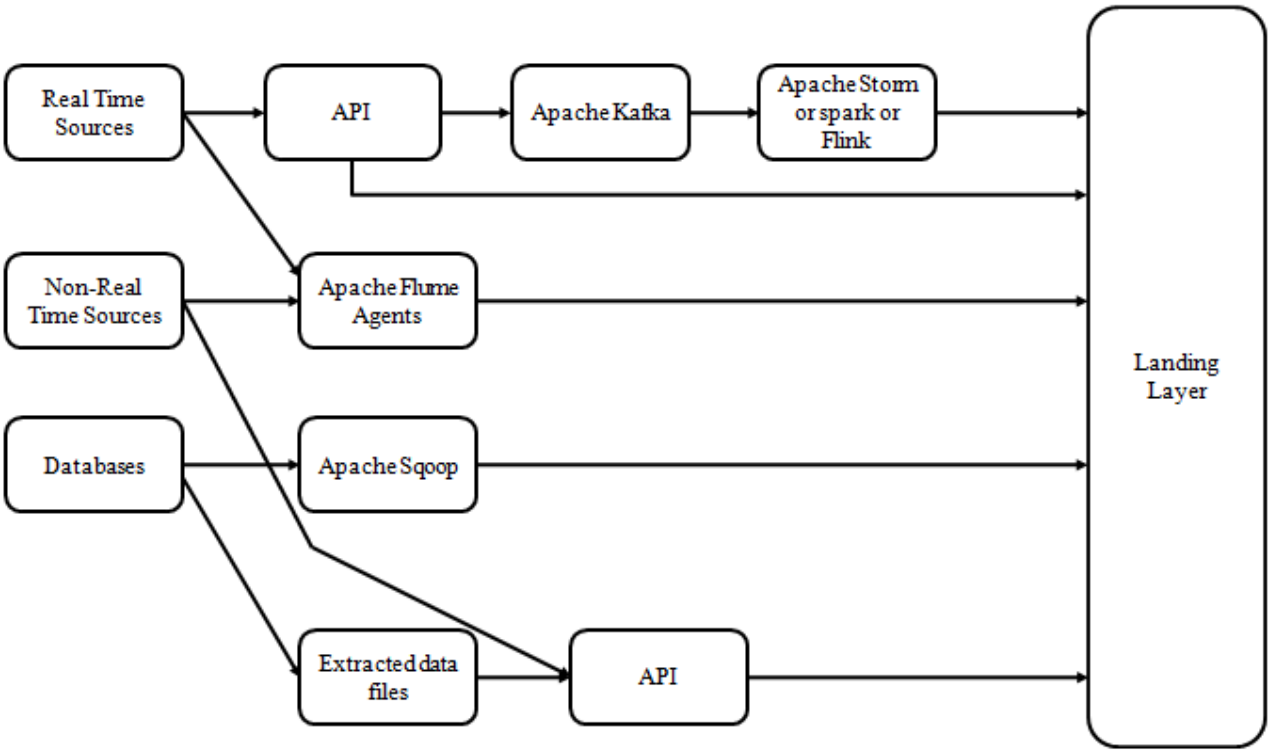

Figure 2: Data Ingestion pattern architecture

3.5. Data Preparation: In this stage data is transformed and prepared for analytical and data processing needs and contains two layer of data storage – structured layer and curated layer. Data from the landing layer is first moved to typed table structures in the structured layer. This layer does the first stage of transformation and provides the benefit of data made available in typed columns instead of strings. Data is then moved to the curated layer where it is stored in a data model format. In this layer all the structured tables are first quality checked and then integrated with various sources. Data is then transformed through various calculations, aggregations and stored into a defined enterprise data model after doing checks for referential integrity. Curated layer is similar to an enterprise data warehouse and only contains data which will be used by consumers.

Data models in big data differ significantly from traditional data warehouses. Driver for the data modeling approach differences in big data comes due to the fact that mutable changes – "UPDATE, APPEND and DELETE" are a complex operation in a big data stack. Due to this dimension, snow flaked dimensions and facts have to be handled differently. Slowly changing dimensions need to turn into snapshot dimensions to eliminate updates. This means every day the entire dimension is re-written into a versioned snapshot or only changed records are written into separate file. The structure of file answers the historical questions easily by joining with a filter on a specific day. Generation of surrogate keys as integer sequences in big data is also a complex operation to parallelize. UUIDs and hashes are easy to parallelize and considered the best approach to generate IDs in big data sets. For snapshot dimensions hashes are better suitable than UUIDs as UUIDs are random where as hash is based on the concatenation of keys that make up the identity of a record and hence can be computed with current data only. In the cases where updates cannot be skipped, utilizing delta lake project is an option. Facts in big data model should be at the lowest level of detail as pre-aggregating them might cause issues in solving analytic requirements from business. Accumulating facts also can be made insert only by inserting a new state that symbolizes the end of the previous states, rather than updating the previous states when record ends. All facts should be partitioned on extraction date and event date in order to create immutable partitions that can be overwritten in case of errors in upstream or business requirement changes.

Due to the data duplication which happens in big data for supporting fault tolerance and data model variations, it is very critical to choose a correct file format. Parquet format works the best for big data platforms. It is a columnar general-purpose storage format designed for Hadoop workloads. Storing data in a columnar format lets the reader read, decompress, and process only the values that are required for a query. Parquet provides multiple benefits which include returning only required data fields, thereby reducing I/O and increasing performance. It is designed to support complex nested data structures and stores full metadata at the end of files making files self-documenting. It uses efficient and

extensible encoding schemas and provides support to work with Avro and Thrift APIs. Parquet provides efficient compression which can be done even on a per-column level and currently supports 3 compression codecs - gzip, snappy and LZO. Snappy provides the best processing performance for compression, decompression; whereas gzip provides best compression performance. Organizations can make the choice between snappy and gzip based on their requirements. Data sets prepared in the structured layer and curated layer are in the parquet format.

Cost of storage is directly proportional to the read, write performance of the storage. Considering volume of data dealt in big data platforms is huge, it is good to make data classifications during the on boarding of an application itself; otherwise the cost of storage could become very high. Data classifications can be broadly classified into three types – hot data, warm data, and cold data. Hot datasets have real time need, heavily accessed and need to be readily available for use. Warm data sets are less frequently accessed and need to be readily available for use. Cold datasets are rarely accessed and SLA's on their availability is tolerant. Multiple storage options are currently available for on premise and cloud infrastructures and based on the data classification requirements the corresponding storage format should be chosen. Search is a critical capability needed for big data platform due to high number of files in the platform. Search will be used for discovery based on metadata and content of the documents stored in BDP. In order to support search, the data needs to be indexed first. To analyze, the document repositories requires a search ability that transforms the millions of unstructured or structured data or documents into useful content which will be useful for BI or advanced analytics. Solr is the preferred search engine for indexing documents due its availability in Hadoop framework and its ability to work on the same cluster as Hadoop. Solr internally utilizes the Apache Lucene libraries.

3.6. Data Insights: This stage comprises of two storage layers called as consumer layer and Analytics layer. Consumer layer provides an easy access point for consumers. This is similar to a data mart where the data from a normalized enterprise data warehouse (curated layer) is de-normalized and made available for business reporting. Consumer layer maintains the data in the HIVE or Impala – Partitioned, external tables for vending. Analytics layer is the area where data scientists analyze and experiment with data in the lake. Data scientists are provided with an access to all data in the big data platform. Access includes looking at data in landing layer, structured layer and curated layer. Benefit of the analytics layer is that data scientists are also allowed to get data from outside the lake. One of the main ways that users will be able to process data and run analytics is through three ways – Python, R and SAS. Main difference of the big data approach for analytics is to "bring processing to data" - processing (function logic) is brought to the data, which is on a node in Hadoop. In traditional data analytics approach data is brought to the analytics server environment and processed there – "Bring data to processing".

Decision on the tool to be used for analytics is dependent on the skill set available and cost. Open Source R and python are free and very good at exposing the data in Hadoop for users to work with, but users must be versed in Mapreduce or Spark in order to execute more advanced processing procedures. SAS on the other hand is proprietary and has an extensive library of functions (including the more complex, difficult Mapreduce implementations) available to the user once LASR Server is incorporate alongside HDFS. Once insights are computed, results are delivered to client applications using SFTP to a FTP server. This guarantees file delivery in a secure fashion and abstracts Hadoop over the cloud. Real-time data that is streamed through the platform will be able to run through Storm or Spark to deliver real-time streaming analytics and alerts derived from the data.

3.7. Metadata: Metadata is used to interact with data through higher level logical abstraction to a table rather than as a mere collection of files on HDFS. Metadata abstracts where data is actually stored in HDFS. Metadata is very critical to a big data platform success. It allows for supplying information about data like partitioning or sorting properties that can be leveraged by various tools. Data management tools can hook into the metadata and allow for data discovery and data lineage analysis. Some of the metadata in Hadoop systems include – document ingestion process metadata, document catalog, Hadoop metadata, data provenance metadata and dataset statistics metadata. **Document ingestion metadata** is passed by source systems that interact with BDP using the landing patterns wherever possible. BDP will provide a library for generating the metadata content. The generated metadata can be passed along with the payload. In cases where metadata content cannot be provided by the business system, BDP ingestion service will maintain the metadata tied to the filename or event. This metadata is maintained within an HBase table. **Document Catalog** is a catalog of data getting ingested to HDFS. The catalog is also stored in HBase.

All the relevant metadata that was passed along with the data payload will be stored in the Document catalog with a link to the file in HDFS. In case of small files that are not used for Hive queries entire data will be stored in the document catalog along with other associated metadata content. This is a pseudo metadata store since there are cases where the actual payload gets stored along with the actual metadata content. **Hadoop Metadata** includes information like permissions and ownership of files and the location of various blocks of that file on data nodes. Such information is usually stored and managed by Hadoop Name Node. HBase stores information like table names, associated namespace, associated attributes and the names of column families. **Data Provenance Metadata** includes information like which user generated a given data set, where the data set came from, how long it took to generate it, and how many records there are or the size of the data loaded. **Dataset statistics Metadata** includes information like the number of

rows in a data set, the number of unique values in each column, a histogram of the distribution of data, and maximum and minimum values. Such metadata is useful for various tools that can leverage it for optimizing execution plans. In addition, data analysts use the metadata to perform analysis. Below provided is a sample data model for the HBASE tables used for document ingestion metadata and document catalog.

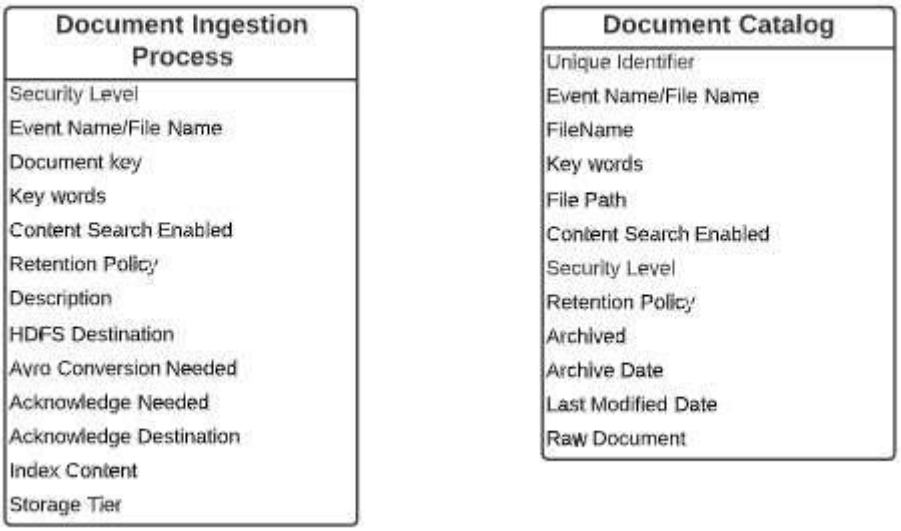

Figure 3: Data model for data ingestion metadata

Description of few of the HBASE Metadata columns has been provided. Security level column contains access restrictions to stored documents – internal/restricted/high. Event Name/File Name contains name of the file or event being processed. Document key contains unique Key for the document. If application does not provide this, then UUID should be used. Keywords contain the search text - comma separated key: value. Content Search Enabled contains if searching is enabled on this document. Default is TRUE. Description contains document description. HDFS destination is the destination of the HDFS folder under /data/landing. Index content contains if the file content need to be indexed using the indexing strategy (Solr/Elastic Search).

4. HDFS Directory Structure:

Defining a proper directory structure helps with providing fine grained access controls and segregation of functions. HDFS follows POSIX standards. With HDFS being a file system, directory structures are part of metadata. Directory structures are used for Organization, "Access control, visibility, and auditing", Resource control and allocation, Partitioning. Directory structures can be organized in the order of data lifecycle, application and age. /data, /user are the life cycle root directories for data and user. /data/landing, /data/structured, /data/curated, /data/consumer, /data/analytics are the base folders for landing, structured, curated, consumer, analytics layers. Underneath the layer base folders enterprises can create application specific folders and similarly under user root directory, user specific folders can be created.

5. Scheduling:

Hadoop ecosystem has a workflow and coordination service for managing Hadoop jobs called as Apache Oozie Workflow Scheduler. Three main components of Oozie are Oozie Workflow jobs, Oozie Coordinator jobs, Oozie Bundle jobs. Oozie Workflow jobs are directed acyclical graphs (DAGs) of actions; actions are typically Hadoop jobs - Mapreduce, Streaming, Pipes, Pig, Hive, Sqoop, etc. Workflow jobs can contain Sub-workflows. Oozie Coordinator jobs trigger recurrent Workflow jobs based on time (frequency) and data availability. Oozie Bundle jobs are sets of Coordinator jobs managed as a single job. Organizations should continue to use the enterprise schedulers like control M or Autosys (when available) instead of utilizing Oozie schedulers, as enterprise scheduler provide the flexibility of placing dependencies between various technologies and applications outside of big data. For example a source system jobs which generate extracts for big data or the database on which Sqoop jobs need to be run can be made predecessors for their respective big data load jobs as they all are under the same umbrella in an enterprise scheduler.

6. Technology: Below table provides the various components of big data platform and the technologies which are recommended:

Table 2: Technology recommendations for big data platform

Function	Storage Layer	Pattern	Framework or Tools	Programming Language
Data Ingestion	Landing	Real Time	Apache Spark/Apache Kafka/Apache Avro	Python
Data Ingestion	Landing	Real Time - Database	Flume/Apache Avro	Python
Data Ingestion	Landing	Batch - Database	SQOOP/Apache Avro	Python
Data Ingestion	Landing	Asynchronous	Raw AS-IS	Python
Data Preparation	Structured	Structured	Apache Spark/Apache Parquet	Python
Data Preparation	Curated	Curated	Apache Spark/ Apache Parquet	Python
Data Insights	Consumer	Consumer	Apache Spark/Hive or Impala	Python
Data Insights	Analytics	Analytics	Apache Spark	Python
Metadata	All	All	HBASE	Python
Scheduler	All	All	Enterprise Schedulers	Python

7. CONCLUSION:

Big data platforms with their complex technologies and native coding requirement need a robust architecture to be successful. With big data platform being extensively used as a replacement for traditional data warehouses it is very critical to have clear data boundaries. In this paper author provides a detailed architecture for "big data platforms with Apache Hadoop tools" which simplifies the complexities of big data processing and help enterprises in building a stable and clean platform. Architecture provided is very robust and can easily be extended to the "big data platforms with cloud tools" by changing the frameworks, tools and storage options to the ones provided by cloud vendors.

REFERENCES:

Journal Papers:

1. Madhusudhan Reddy Sureddy, Prathyusha Yallamula (2020), DATA QUALITY ARCHITECTURE FOR DATA WAREHOUSES, International Journal of Research Cultural Society , 4(6), pp 95-100
2. Madhusudhan Reddy Sureddy, Prathyusha Yallamula (2020), A Framework for Monitoring Data Warehousing Applications, International Research Journal of Engineering and Technology, 7(6), pp 7023-7029
3. Madhusudhan Reddy Sureddy, Prathyusha Yallamula (2020). Approach to help choose right data warehousing tool for an enterprise. International Journal of Advance Research, Ideas and Innovations in Technology, 6(4) www.IJARIIT.com.

Proceedings Papers: (11)

1. Madhusudhan Reddy Sureddy, Prathyusha Yallamula (2020), DEBUGGING METHODOLOGY FOR ISSUES IN DATA WAREHOUSING AND BUSINESS INTELLIGENCE PLATFORMS, Proceedings of IFERP International conference, Luxor, Egypt, August 10-11, 2020

Web References:

- https://www.geeksforgeeks.org/5-vs-of-big-data/
- https://www.8bitmen.com/what-is-data-ingestion-how-to-pick-the-right-data-ingestion-tool/
- https://wire19.com/big-data-processing-services-comparison
- https://www.xplenty.com/blog/storing-apache-hadoop-data-cloud-hdfs-vs-s3/
- https://www.neosllc.com/the-five-zones-every-data-lake-should-consider/
- https://www.scnsoft.com/blog/data-lake-implementation-approaches
- https://www.waitingforcode.com/apache-parquet/compression-parquet/read
- https://acadgild.com/blog/parquet-file-format-hadoop
- https://www.dataversity.net/brief-history-big-data/
- https://www.weforum.org/agenda/2015/02/a-brief-history-of-big-data-everyone-should-read/
- https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf

- https://en.wikipedia.org/wiki/Apache_Hadoop
- https://www.datasciencecentral.com/profiles/blogs/how-many-v-s-in-big-data-the-characteristics-that-define-big-data
- https://mindmajix.com/cloudera-vs-hortonworks
- https://www.edureka.co/blog/hadoop-ecosystem
- https://authenticredcreative.com/characteristics-of-big-data-the-3-differentiating-v
- https://www.datacouncil.ai/blog/functional-data-engineering-a-modern-paradigm-for-batch-data-processing
- https://delta.io/
- https://www.oreilly.com/library/view/hadoop-application-architectures/9781491910313/ch01.html
- https://dataconomy.com/2016/10/big-data-python/

Author Biography

I am Madhusudhan Reddy Sureddy with qualifications of Bachelors of engineering from Osmania University. I have 14 years of Information Technology experience and worked with fortune 500 companies - GE Capital Americas, American Red Cross, CVS Caremark, FannieMae and Santander Bank and in the business domains BFSI (Banking, Financial Services and Insurance), Mortgage and Healthcare. My research fields include data warehousing, data integration, data quality, data virtualization, master data management, reference data management, metadata management, big data solutions, business rule engines, data modelling, data analytics, Operational and Support Model, server management and Capacity planning.