Hadoop Proof of Concept Document 2

Neil Bardhan

Business Problem

Currently, clinical data are stored in a siloed, server based system. It is difficult or impossible to search across studies. Prior knowledge of data location is necessary for any study integration needs.

What are the pros/cons of using Hadoop to store clinical data vs current server-based system?

PROs

Hadoop offers HDFS, a distributed file storage system where data is split into manageable chunks of fixed sizes (64 MB or 128 MB) and stored "on the cloud", which basically means that they are stored on low-cost commodity hardware with less emphasis on storage and greater emphasis on availability. Each chunk is replicated three times and each of the three copies are stored on different machines called DataNodes. All the details required to recreate the original datset are stored on a DataNode which has been given admin status, called the NameNode. The main advantages of Hadoop are that -

1. **Hadoop (HDFS) does not care what format your data is in.** It could be text, video, audio, log files, DBMS tables, anything.

2. **There are no storage or space constraints in HDFS.** Unlike servers which have storage limits (because they are a single machine), HDFS has no storage limits. You can store as much data as you want from as many varied data sources.

3. **Hadoop takes care of the data storage locations** unlike server based systems where we must store the data in folders and directories. Similarly when we need that data for analytics, we must know where (filepath) it is stored. Hadoop frees us from the trouble so that we concentrate on the analytics.

4. **Hadoop is scaleable.** As mentioned previously, HDFS never runs out of space (well it does but you can always get more in **seconds**). This is not the case with servers. When servers run out of memory, they need to be reconfigured (Hardware needs to be updated, etc.). This can take a long time and potentially eat into time that could have been invested in analysis.

5. **Hadoop has a very powerful ecosystem.** Traditionally, MapReduce has been considered the processing workhorse of Hadoop. However, there are a lot of open-sourced tools aimed at all kinds of data professionals from SQL like tools to processing frameworks like Apache Spark with support for Machine Learning Algorithms and APIs for popular languages like Python and R.

CONs

1. **Hadoop as well as Spark cannot handle small files very well,** and from a clincal data point of view that is a major problem. My analysis with Adverse Event data on Databricks using a 6GB cluster took significantly longer than the same analysis performed on my local machine. Any file smaller than the HDFS block size of 64 MB faces this problem. A huge number of small files generate a lot of metadata which puts a lot of load on the namenode. This is a major roadblock when it comes to clincal trial data where the files are smaller than 64 MB, however, Hadoop as well as Spark would be great

for procesing and storing biomarker and health sensor data which are much larger and therefore optimal for Spark and Hadoop.

2. **Security** - The use of a Hadoop ecosystem in an organization ensures that data that was once siloed away securely is now brought into a vast data lake to facilitate ease of pre-processing and analytics. This leads to new security challenges. Security challenges are mainly of two types **internal** and **external**.
   Internal security challenges are ensuring *that the right people have access to the right data.* Ensuring that only authorized users can access the data they are entitled to access **and** keeping track (data access histories) for all users are major internal security challenges.
   External security challenges involve insuring that the data cannot be accessed by malicious entities from outside the system. External security threats are larger considering the sensitivity of the data. Usually, cloud service providers like Cloudera provide external data security in their enterprise editions and it can be highly configured to suit the organization's needs. The advantage of storing data on servers is that the organization gets to dictate the security policy in its entirety. However as shown by the recent Anthem Medical Data Breach last year, the risks of storing data on one's own servers without adequate protection are enormous. There is very little literature comparing the efficacy of securely storing data on the cloud versus securely storing data on the server with regards to **external** security threats.

3. **Learning Curve and volatility** - Considering that Hadoop or Distributed Parallel Processing is a relatively new concept, getting a proper, in-depth understanding of "Hadoop" is quite time consuming and involves a slight learning curve. There are so many tools and so much jargon associated with the Hadoop ecosystem that it is quite easy to get confused about which tool best suits one's purpose. Another problem with using a Hadoop ecosystem is the volatility of technologies. Every couple of months there is a radical update that requires changing the legacy code to make the system compatible with the update and this is a cumbersome, time-consuming task. Also every couple of years a new "game-changing" tool (like Spark for example) comes along and it makes more sense to migrate to that tool. In order to do that, the system must be reconfigured and that again eats into time that could have been invested in data analysis.

What are the steps needed to ingest and profile data in Hadoop to make data recognizable in Solr search? Is there an alternative to searching via Solr?

1. **Indexing Data into Solr** - To be able to search a structured or semi-structured dataset, Solr needs to have an index of the schema of that dataset. Solr can accept data from many different sources, including XML, CSV, RDBMS Tables, JSON files and unstructured formats like DOC, TXT and PDF. The Solr index is a XML file where we specify the schema of the data to be searched. The XML file is named **schema.xml** and is located in the **/tmp** directory. To be able to search across studies as is the stated requirement, we must create and load schemas for each field in each CSV file. This process will have to be automated (most probably using a Python script) such that every time a new study is loaded, fields that are not in the index get added onto the index. Consider the snippet of AE data -

   The corresponding XML Schema will look like this -

RANDDT	RAND	TXFDT	EVALS	EVALE	AGE	ETHNIC	IL13SIG	PERIOS
9/23/2009	YES	9/23/2009	YES	YES	64	Not Hispanic or Latino	POSITIVE	HIGH
12/9/2009	YES	12/9/2009	YES	YES	57	Not Hispanic or Latino	POSITIVE	HIGH
12/8/2009	YES	12/8/2009	YES	YES	61	Not Hispanic or Latino	POSITIVE	HIGH
1/23/2010	YES	1/23/2010	YES	YES	62	Not Hispanic or Latino	NEGATIVE	HIGH
1/23/2010	YES	1/23/2010	YES	YES	62	Not Hispanic or Latino	NEGATIVE	HIGH
7/21/2009	YES	7/21/2009	YES	YES	40	Not Hispanic or Latino	POSITIVE	LOW
7/21/2009	YES	7/21/2009	YES	YES	40	Not Hispanic or Latino	POSITIVE	LOW
7/21/2009	YES	7/21/2009	YES	YES	40	Not Hispanic or Latino	POSITIVE	LOW

Figure 1: Snippet of AE data.

```
<field name="RANDDT" type="date" indexed="true" stored="true" required="true"/>
<field name="RAND" type="string" indexed="false" stored="true"/>
<field name="TXFDT" type="date" indexed="true" stored="true" omitNorms="true"/>
<field name="EVALS" type="string" indexed="true" stored="true" omitNorms="true"/>
<field name="EVALE" type="string" indexed="false" stored="true" omitNorms="true"/>
<field name="AGE" type="int" indexed="false" stored="true" omitNorms="true"/>
<field name="ETHNIC" type="string" indexed="true" stored="true"/>
<field name=" IL13SIG" type="string" indexed="true" stored="true"/>
<field name="PERIOS" type="string" indexed="true" stored="true"/>
```

Figure 2: XML schema for the above snippet of AE data.

Solr search is ready to use once the schema is properly loaded. Solr search is very powerful and supports the use of regular expressions to narrow down the search results.

2. **Alternatives to Solr** - There are two mainstream alternatives to Solr. **ElasticSearch** and **Sphinx**.
   ElasticSearch is a powerful graph based search engine used by Facebook, Netflix and StackExchange to find relations and links within the data as well as full text search. It is based on a similar principle as Solr and shares the same underlying search engine (Apache Lucene). It can be used for all kinds of documents and provides scalable, real time search. However, since most Hadoop distributions, especially Cloudera use Solr, I don't think ElasticSearch can be used with Hadoop, it will have to be used in Standalone mode.
   Sphinx is an open-source text search engine geared to provide search functionality to highly structure DBMS/SQL tables. Sphinx works with fixed schema, that is, a set of predefined attribute columns. Thus, automatically updating the index will be a cumbersome task. Also since Sphinx deals with SQL tables, it doesn't work well with columns that have very little data. To overcome the above problems, we will have to convert our data to JSON format (sas7bdat to CSV to JSON). Sphinx will also be have to used in Standalone mode as Cloudera support is limited to Solr.

If RWD are stored on Teradata, and biomarker and clinical data are archived in HDFS, how feasible is it to use a connector to read selected clinical/biomarker data into Teradata for integration and further analysis?

As I have understood from previous meetings -

- Real World Data is **highly structured**.

- Teradata is being used for data warehousing purposes.

Therefore, I am making the assumption that the highly structured real world data will be stored in the Teradata Data Warehouse. Any data that is stored in a data warehouse can be expected to be highly structured and extremely *clean*. Meaning, that there can be **no empty cells** because empty cells jeopardize the integrity of the DBMS table.
Teradata offers connectors for both Spark and Hadoop. There is a major difference in what purposes the connectors serve. The Hadoop-Teradata connector provides high-speed, bulk transfers for large RDBMS tables. The Hadoop connector enables **bi-directional transfer of data** from HDFS to Teradata's Data Warehouse. Usually, HDFS serves as a pre-processing stage where large datasets are cleaned and normalized (Using Hive or Impala) before being loaded into Teradata via the connector. Once in Teradata, the data tables can be used for OLAP (Online Analytical Processing) and other decision support systems.
The **Teradata-Spark** connector on the other hand, is also bi-directional connector but is most useful when we need to run Non-SQL, Machine Learning algorithms or even simple Non-SQL analytics on the data. In this case we transfer data from Teradata to Spark, we perform the analysis on Spark, format the results to Teradata friendly tables in Spark and then return the results to Teradata to be stored for future reference.
The following diagram highlights Hadoop and Spark connectivity to Teradata and the most likely use cases.

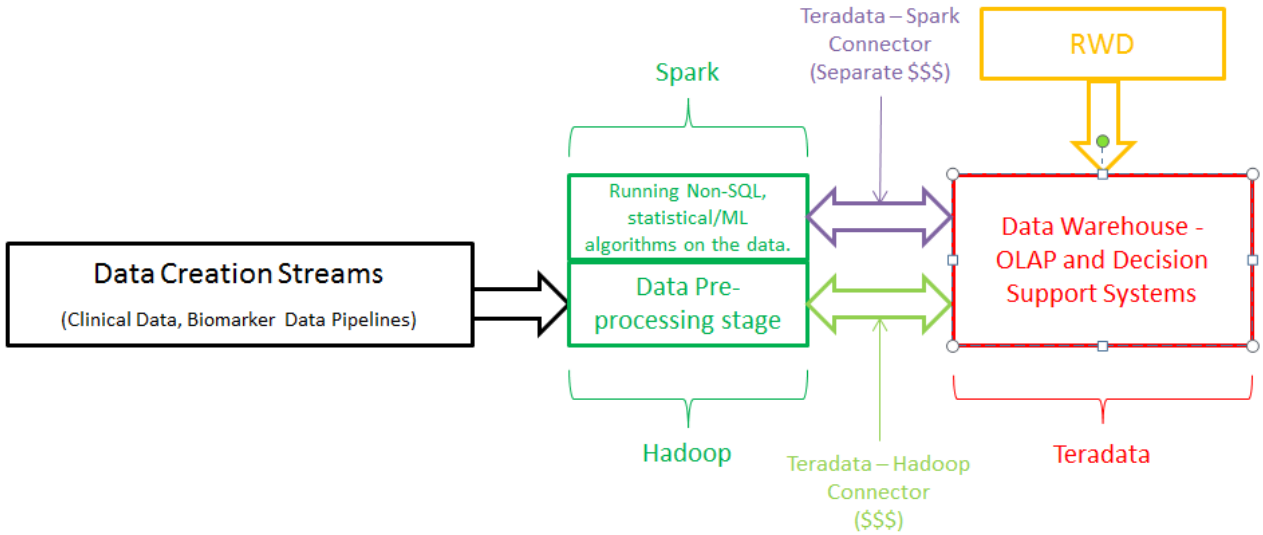

Figure 3: Teradata-Hadoop-Spark Pipeline

What are the advantages of using the analytics capabilities available within the Hadoop ecosystem (Spark, any others?) vs other tools such as Teradata, R Studio, Jupyter, etc?

Note that data can be transferred from Teradata to Hadoop for analytics as well. Hadoop (being open-sourced) can be configured to internally transfer the data to Spark. Spark can then perform the analysis and return the results to Hadoop which in turn can format the results into SQL tables and return them to Teradata. That would reduce the need to buy a separate Spark connector, but the Hadoop ecosystem will have to be internally configured to facilitate this. This is perhaps an optimal way of using the Hadoop ecosystem alongside Ter-

adata. It may take the data analysis process a little longer than if you were to use the Spark and Hadoop connectors separately, but it would help us avoid paying for the Spark-Teradata connector.
Given that Teradata will be used, certain analytic capabilities available within Hadoop will be rendered less useful. Considering that Teradata has highly advanced SQL tools in it's suite, it would be redundant to use Hive and Impala (both SQL based tools) in Hadoop *for analysis jobs*. Note that these tools along with Spark will be very useful in **transforming** non-relational data to SQL tables before loading those tables into Teradata.
One problem with this setup could be the variability of the data from study to study. I am assuming that Real World Data is **highly standardized** and has little or no variability. Consider that we have a SQL table in Teradata as follows -

ARM	ARMCD	AELLT	AESER	AESEV	AESHOSP	PHASE
Lebrikizumab 125mg	L125	Exacerbation of Asthma	Non-Serious	Mild	N	Phase 1
Lebrikizumab 125mg	L125	Injection Site Reaction	Non-Serious	Moderate	Y	Phase 2
Lebrikizumab 125mg	L125	Exacerbation of Asthma	Serious	Severe	Y	Phase 2

Figure 4: Table 1 - SQL Table in Teradata

and we have a SQL table in Hadoop as follows -

ARM	ARMCD	AELLT	AESER	AESEV	AESHOSP
Lebrikizumab 37.5mg	L37.5	Injection Site Reaction	Non-Serious	Mild	N
Lebrikizumab 125mg	L125	Exacerbation of Asthma	Non-Serious	Moderate	Y
Lebrikizumab 250mg	L250	Injection Site Reaction	Serious	Severe	Y

Figure 5: Table 2 - SQL Table in Hadoop

When we import Table 2 into Teradata, we are faced with the problem of joining these two tables for comprehensive analysis. Joining these two tables as is would lead to the introduction of NULL values for the corresponding values in Table 2. A workaround to avoid NULL values would be drop the PHASE column from table 1 before joining, but that would simply make that column redundant for future use. A lengthier workaround would be transfer both tables to Hadoop, *simulate* the missing values and return the joined table to Teradata for further analysis.
An important question here is whether Spark has any analytical advantages over currently used analytics tools from a **clinical trial data perspective**. The short answer is **no**. Clinical trial data have sizes ranging from 10 KB to 600 MB and that is very small from a Spark point of view (except for lb and cm files which are perfect). Spark and Hadoop in general find it difficult to handle small files due to reasons mentioned previously. In the time it takes to load the small data into Spark one can begin and finish with the analysis of that same data on the local RStudio or on r.roche.com. The main advantage of using Spark is it's scalability. Consider only the lb data for example. Performing data analysis on lb data on the local machine will take much longer than it will on Spark. As data sizes increase, so does the speed at which Spark performs analytics. In fact the speed at which Spark performs analytics increases at an

exponential rate with linear increase in data size. Even though I haven't had a chance to look at biomarker data, I would presume that biomarker data as well as health sensor data for PHC 2.0 could be a perfect fit for Spark rather than clinical data. From a data storage perspective, any kind of data, regardless of size can be stored in HDFS. Storage should not be a problem at all.