# SQL-Like Big Data Environments: Case study in Clinical Trial Analytics

Akshay Grover[1], Jay Gholap[2], Vandana P. Janeja[2], Yelena Yesha[1]
[1]Computer Science and Electrical Engineering,
[2]Information Systems
University of Maryland, Baltimore County
{akshay2, jgholap1,vjaneja, yeyesha}@umbc.edu

Raghu Chintalapati[3], Harsh Marwaha[3], Kunal Modi[3]
[3]Ekagra Software Technologies
{Raghu.Chintalapati, Harsh.Marwaha, Kunal.Modi}@ekagra.com

**Abstract- Big Data deals with enormous volumes of complex and exponentially growing data sets from multiple sources. With rapid growth in technology, we are now able to generate immense amount of data in almost any field imaginable including physical, biological and biomedical sciences. With the diversity and amount of data in health care industry there is an increasing need to evaluate the components in big data frameworks and gauge their adaptability to analytics techniques. However, a key step in adapting big data tools is the portability of relational databases to big data environment. Since SQL is considered to be the de-facto language for interactive queries, in this paper, we evaluate the performance of SQL-like big data solutions for the portability of existing relational databases. Our work focuses on benchmarking multiple SQL-like big data technologies over Hadoop based distributed file system (HDFS) for Study Data Tabulation Model (SDTM) used in clinical trial databases for improving the efficiency of research in clinical trials. We use publically available clinical trial data (from National Institute on Drug Abuse (NIDA)), which follows SDTM, as a test bed to measure key parameters like usability, adaptability, modularity, robustness and efficiency of these solutions. With the intention to demonstrate how current clinical trial functionality can be replicated on a big data backend with high SQL-like functionality, we evaluate several types of ad-hoc SQL queries.**

***Keywords- Big Data; Benchmarking; HDFS; SDTM; SQL like***

## I. INTRODUCTION

The focus of this paper is the evaluation of SQL-like Big Data Environment for clinical trials data. We set the stage by explaining what are clinical trials and what data models are currently being followed. We go a step beyond and explain the need of Big data environment and particularly SQL-like Big Data environment in the domain of clinical trials.

Clinical trials as defined by NIH (National Institutes of Health) [1] are the research studies in which one or more subjects are prospectively given one or more interventions to evaluate their effects on specific outcomes such as health-related behavioral or biomedical outcomes. ClinicalTrials.gov, run by the United States National Library of Medicine (NLM) is the first and the largest clinical trials registry (CTR) for registering a clinical trial. ClinicalTrials.gov currently lists over 199,000 studies with locations across50 states in the United States and in 190 countries worldwide [17].

The United States Food and Drug Administration (FDA), on July 21, 2004, [2] selected Study Data Tabulation Model (SDTM) as the standard requirement for submitting the tabulation data to FDA for clinical trials. The model defines a standard structure for both the human clinical trial (study) data tabulations and the for nonclinical study data tabulations. It provides a generalized framework for describing the organization of information collected during human and animal trials. The SDTM is built around the concept of observations , which consist of discrete pieces of information gathered during a study. Observations are reported in series of domains, usually corresponding to data that are collected together.

With the advent of the era for personalized medicine, the number of clinical trials has grown exponentially. The data is being collected from combination of several smart devices, which is further improving the design of clinical trials, their efficiency and outcomes. Electronic data capture is helping in recording patient information in the provider's electronic medical records and using such electronic medical records as the primary source for clinical-trial data is accelerating the trials and hence reducing the likelihood of errors caused by manual or duplicate entries. We can leverage the capabilities of big data technologies to unravel the hidden correlations such as drug interaction, comorbidity etc. across several clinical trials.

According to literature collected from clinicaltrials.gov, most of the clinical trials today call for the enrollment of 1 in every 200 Americans as study participant. With such a large cohort of population signing for Clinical Trials and thus generating huge volume of data, one should ask the question if this high level of human participation is being put to the best use possible. Most clinical trials include paper files, X-

ray films, patient narratives, doctor's prescription, etc. thus accounting towards the variety of data. Velocity of mounting data increases with data that represents routine monitoring, such as blood pressure readings or several diabetic glucose measurements. With all the 3 Vs in place i.e. Volume, Velocity, and Variety - Big Data promises making the drug discovery and development process more efficient in case of Clinical Trials.

Big Data in general refers to very large and complex datasets such that the data becomes unmanageable via traditional database management tools. With SQL being the default choice and by far the best-suited language for basic data analysis already extensively in use in the domain of clinical trials, we have a number of big data solutions leveraging the same. Solutions like Hive, Presto, Impala, and Shark are designed to support data analytics on big data with a SQL like support so that transition into the big data environment is seamless.

Thus, in the following sections of the paper, we evaluate the portability of SDTM relational model to SQL like big data environment and understand the various challenges involved during the process of ELT (Extract, Load and Transform) from various source systems. We also propose solutions to address the issue of generating surrogate keys, by incorporating cryptographic algorithms like SHA 256, to maintain the data and referential integrity. Finally, we perform extensive evaluations across multiple SQL like big data environments and propose our overall observations.

The paper is organized as follows: In section II we discuss the related work. Section III discusses the state of the art of the big data solution landscape. In section IV we outline our methodology and in section V we discuss the results. Section VI outlines the overall observations and discussions followed by conclusions and future work in section VII.

## II. Related Work

Recent advances in healthcare and medical devices have produced massive amounts of multimodal data and hence the need for parallel processing is apparent for mining these data sets, which can range anywhere from tens of gigabytes, to terabytes or even petabytes. The emergence of such massive datasets in clinical settings, presents both challenges and opportunities in data storage and analysis. Here we discuss such challenges and related work:

### *A. Surrogate Keys*

Identifiers are the basic structure in every modern database implementation and the core of every relational database ensuring the consistence of data and relational paradigm using primary and foreign keys for schemas and object reference. The importance of uniqueness of identifiers is crucial for efficient data retrieval, storage and comparison in a distributed system.

Balkić et al.[23] address the issue of distributed data storage and widely used unique identifiers which are in use from the early use of databases and backend systems. They represent a new method for objects tagging using synergy mechanism between well-known Geo hash algorithm and Universally Unique Identifiers called GHUUID. Such keys can be generated by converting referential spatial systems into the Geo datum enabled WGS84 or country specific geo datum.

Hunt[24] incorporates UUID (Leach, Mealling & Salz, 2005) into a database design, as a type of artificial data that can be used as primary key to synchronize the data between two or more computers via an email server. The approach discussed by [24] proves the use of UUID to be robust and quiet successful for synchronizing data across multiple installations.

Iordanov [25] presents a novel graph database based on generalized hypergraphs where hyperedges contained other hyperedges. The approach discussed by [25] maps the identifiers to either a tuple of identifiers or to a plain byte array. They concluded that use of UUID made it easier to manage the identifiers in a large distributed environment while virtually eliminating the chance of collision and each data peer could make up new IDs without a central authority.

Our work addresses the problem of surrogate keys in the domain of clinical trials. Application of Geo Hash Algorithm proposed in [23] is tedious to implement in a big data environment. [24] and [25] implement plain UUID in their design. We incorporate the use of cryptographic algorithms such as SHA 256 on UUIDs as another approach to generate practically unique surrogate keys in the big data environment and hence maintain the referential integrity of the data.

### *B. ELT (Extract, Load and Transform) in Big Data*

Understanding the nature of how data warehouses are loaded and the data movement tools work, can be really helpful in analytics of the data.

ETL (Extract, Load and Transform) refers to the process of moving data from source systems into a data warehouse. The data is:

- Copied from the source system to the staging area (Extracted).
- Reformatted for the warehouse with business calculations applied (Transform)
- Copied from the staging area into the warehouse (Load)

In the modern era big data engines support complex transformations in SQL, including in-database data mining, cleaning, validation, statistical algorithms, profiling, drilldown functionality and much more. It is more efficient to perform most types of transformation within this engine. Hence, a new approach emerged where; data is extracted from the sources, loaded into staging tables and then transformed into desired format. This approach is known as ELT (Extract, Load and Transform)

Devi et al. [26] propose process of ELT for taking business intelligence decisions in Apache Hadoop by

performing ELTL. The discussed approach has two major advantages. Firstly, one can ingest massive amounts of data without specifying a schema on write. Secondly, offload the transformation of raw data by parallel processing. Once the data is in Hadoop (HDFS), tasks of cleaning, normalizing, standardizing the data for analytics using MapReduce can be performed.

We propose the novel approach of applying this work in the domain of clinical trials while evaluating the portability of SDTM model on big data backend.

### C. SQL on Hadoop

A significant amount of work has been done in the field of evaluating and benchmarking the tools that provide SQL like functionality over Hadoop. Since SQL is considered to be the de-facto language for data analytics, SQL query processing over Hadoop data has recently gained significant foot. Several enterprise data management tools on hadoop rely on SQL.

Floratou et al.[27] benchmark and evaluate the performance of Hive and Impala by providing TPC-H like workload. The queries discussed in [27] scans the table, applies an inequality predicate, projects a few columns, performs an aggregation and sorts the final result.

Cloudera [28] addresses the implementation and working of Impala, which is a state-of-the art MPP SQL query engine, designed specifically to leverage the flexibility and scalability of Hadoop. It demonstrates that it is possible to build an analytic DBMS on top of Hadoop that performs just as well or better than commercial solutions for RDBMS, but at the same time retains the flexibility and cost-effectiveness of Hadoop.

Our work addresses the issue of benchmarking the leading SQL like tools available for Big Data Environment for their efficiency and efficacy in the domain of Clinical Trials. These tools are currently being used by several organizations to churn petabytes of data. Presto is used to address the ad hoc interactive use cases for data exploration at Netflix [13], while Facebook, VideoEgg and Scribd uses Hive extensively to store and retrieve the data. We explore these SQL like processing platforms by executing the queries similar to the ones executed by Pavlo et al. [3].

Next we discuss the state of the art technologies available as shown in figure 1, some of which can be viable solutions for the clinical trials domain.

## III. STATE OF THE ART IN SQL SUPPORTING BIG DATA TOOLS

Figure 1 gives out a referential list of solutions that form the major solutions available in the big data ecosystem. In our ecosystem review we have branched out tools based on distributed programming, distributed file system, SQL like processing tools etc. These tools can be used as per the needs of the use case.

In this paper we evaluate key big data solutions for their ability to replicate the current clinical trial repository on a big data backend. Of the several big data analytics frameworks present in the market, we target the ones, which impose MPP (Massively Parallel Processing)-like execution engines on top of Hadoop and have high SQL like functionality and portability.

Due to the compelling need for utilizing a SQL like backend in porting the SDTM, we evaluate the following four SQL like systems to provide quantitative and qualitative comparisons:

- Apache Hive
- Facebook Presto
- Apache Drill
- Apache Spark

### A. Apache Hive

Hive [5] is an open-source data warehouse platform, which provides SQL like interface to access data residing in Hadoop distributed file system (HDFS). Developed by Facebook, Hive was primarily built in order to process a large amount of data in Hadoop when it had no alternative other than mapreduce scripts. Hive significantly reduced the complexity to manipulate the enormous data in Hadoop by eliminating the need of writing complex mapreduce tasks. Hive supports SQL-style query language known as HiveQL. Users can easily write HiveQL queries to collect and analyze data for various purposes such as business intelligence, data summarization or interactive data mining. Hive translates these queries into mapreduce jobs and submits them to Hadoop for execution. Hive also allows to plug-in custom user defined functions (UDFs) and aggregation functions (UDAFs) written in Java to perform operations that are not supported by HiveQL. Current version of HiveQL offers only a subset of SQL statements. Although earlier versions of Hive have demonstrated a significant performance enhancement in Hadoop, future versions would include a new set of SQL commands, more efficient query optimization engine and better JDBC and ODBC drivers for smooth integration with third party BI tools.


Figure 1: State of the Art in Big Data Technologies

### B. *Facebook Presto*

Significantly improving the scalability of SQL, Facebook released Presto [6], an open source distributed SQL query engine for running interactive queries on petabyte-sized data-store. Presto was not only designed to query data residing in HDFS but also to query other data sources including relational databases and non-relational databases such as Cassandra. Presto uses in-memory, pipelined processing for execution of queries rather than mapreduce, hence sidesteps unnecessary I/O and associated latency. As a result, Facebook claims that Presto runs 10 times better than hive and Hadoop in terms of CPU efficiency and latency. Facebook built hive in order to give Hadoop some data warehouse and SQL-like capabilities, but since it relies on mapreduce, scanning over a large dataset can take many minutes to hours. It is not an ideal scenario where we need answers on the fly.

Presto supports a larger subset of ANSI SQL including joins, aggregations, subqueries & window functions. Apart from these, it also supports wide range of utility functions including JSON functions, URL functions, string functions and regular expression functions. Presto is turning out to be a popular choice for interactive analysis over the big data because of its support for various data- sources and volume of data it can handle. With Presto, simple queries can get executed in a few hundred milliseconds, while more complex ones execute in a few minutes. It runs in memory and never writes to the disk. Future releases of Presto would target better performance by utilizing data caching to improve query speed.

### C. *Apache Drill*

Apache Drill [7] is a distributed system for interactive ad-hoc analysis of large-scale dataset. Drill is the open source version of Google's Drexel system, which is available as an infrastructure service called Google BigQuery. Developed with the goal to provide low latency and faster interactive queries for larger datasets, it supports several backing stores like HDFS and HBASE etc. Besides the knowledge of ANSI SQL to write and execute the queries, Drill does not require the user to specify the schema before querying. This schema-on-the-fly feature calls for less involvement from IT personnel.

Apache Drill provides ANSI SQL interface for querying data from Hadoop as well as NoSQL databases. Drill supports several file formats such as CSV, TSV, Parquet, JSON and PSV. In distributed setup, data locality based execution speeds up Drill's query processing by reducing network traffic. A variety of SQL functions supported by Apache Drill including mathematical & statistical functions, string and date manipulation functions make it a very powerful analytical tool. Similar to Hive, Drill allows you to create custom functions by writing a custom java code. It also offers an excellent connectivity to business intelligence tools such as Tableau, Microstrategy, Qlikview and Tibco Spotfire. With simple installation, it can scale up-to a very large cluster comprised of thousands of nodes.

### D. *Apache Spark*

Spark provides highly advanced DAG execution engine that supports cyclic data flow and in-memory computing. Spark offers resilient distributed datasets (RDDs) which is an an abstraction to support multiple distinct jobs, each of which reads data from stable storage. RDDs allow Spark to outperform existing solutions by up to 100x in multi-pass analytics. Spark SQL is a fairly new module in Apache Spark that combines relational processing with Spark's functional programming API. Spark SQL offers much tighter integration between relational and procedural processing, through a DataFrame API that combines with procedural Spark code. Secondly, it includes a highly extensible optimizer which is built using features of the Scala, that makes it easy to add composable rules, control code generation, and define extension points.

## IV. STUDY METHODOLOGY

Although Apache Hive, Facebook Presto and Apache Drill provide similar functionality, underlying query execution engines utilized by these tools differ. These software tools provide an interface to process large scale data with scalability up-to thousand distributed nodes, fault tolerance, support for custom pluggable user defined functions. To evaluate these tools and address the relevant challenges of porting data into a big data environment we provide the overall view of our approach as shown in figure 2. The key tasks include extracting, loading and transforming the data, followed by generating surrogate keys so the data records are unique through all mapreduce operations and finally performing queries which are evaluated for their efficiency of execution across all the systems being evaluated.

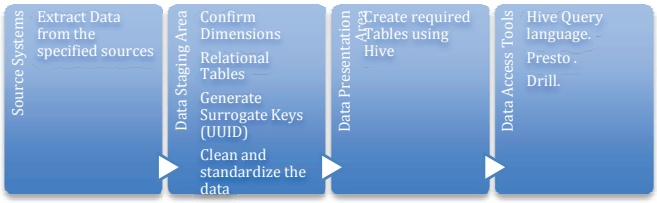


Figure 2: Overall Approach

### A. *ELT (Extract, Load and Transform)*

Inspired by Kimball Architecture [16], we use the Extract-load-transform (ELT) approach to design our evaluation on the big data backend. We extract the data from the specified sources and push the same for batch processing into staging tables. Once we have the data in staging area, we confirm the dimensions of the tables and generate surrogate keys for the smooth integration of data from multiple source systems. Thus, we clean and standardize the data in this stage. Since our ELT approach utilizes intermediate data store before transforming the data, it is much more efficient to handle big volume data than

traditional Extract-Transform-Load (ETL). Once we have the data in temporary storage unit i.e. staging tables, we transform and process the data according to the business rules and push it into the final tables. The processing of data includes the generation of surrogate keys to maintain the data integrity as we discuss next.

*B. Generating Surrogate Keys.*

In order to generate unique keys for the smooth integration of data from multiple source systems we propose three different methods.

- Generate domain specific CTR Key (Numerical Key)
- Generate key using Java UUID
- Generate key using SHA256(Java UUID)

*1) Domain Specific CTR Key (Numerical Key)*

This key utilizes the Clinical Trial repository data to generate the unique keys. The rowid UDF (User Defined Function) provided by hivemall jar i.e. the scalable machine learning library for hive, generates a unique rowid by concatenating TaskID with a sequence which gets incremented by 1 for every iteration for a single insert query. It starts with 0 for every insert query.

We can further increase the granularity of the keys by concatenating the SDTM's StudyID with the rowid function and cast the output as an integer and store it in tables.

*2) Key using Java UUID:Universally Unique Identifier*

Since Hive runs several mappers-reducers in parallel, there is no way to generate a globally unique increasing row id. Generating sequential numbers is also not possible due to the parallel nature of Hadoop. The second option we explored is to generate the Java Unique Identifier, which represents a 128-bit value. According to Oracle there are four different basic types of UUIDs: time-based, DCE (Distributed Computing Environment) security, name-based, and randomly generated UUIDs. A UUID fulfill the requirement of generating unique keys where unique means "practically unique" rather than "guaranteed unique". According to ITU-T [8] after generating 1 billion UUIDs every second for the next 100 years, the probability of creating just one duplicate would be about 50%.

Based on the requirement and data size we can generate and concat two randomly generated Java UUIDs that could serve as the surrogate keys. Although, based on the aforementioned probability of generating a duplicate key, one might never require this approach. Nonetheless, this approach can be summarized as shown in figure 3:

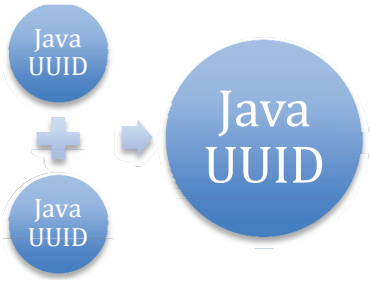


Figure 3: JAVA UUID

*3) Key using SHA256 (Java UUID)*

SHA-256 is one of the successor hash functions to SHA-1 (collectively referred to as SHA-2), and is one of the strongest hash functions available. Using hash (of a key column) can generate collisions. The hash value and its original value are not in a one-to-one mapping. Hence, it is impossible to track the original value from its hash value. SHA1 has collision in theory, but don't exist for strings of short length. Git [9] uses SHA1 hashes as IDs and there are *still* no known SHA1 collisions in 2014. We generate a random string using java UUID or a randomly generated string of binary numbers and pass it through SHA 256 hash function thus generating 256-bit checksum and use the same to serve the purpose for surrogate keys. This approach can be summarized as shown in figure 4:

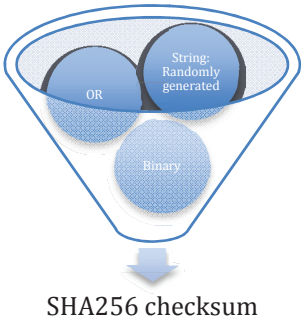


Figure 4: SHA 256

*C. Query Execution*

Once we have the table structure imitating the relational databases in the big data backend, we can then use several SQL query engines present in big data ecosystem to query large distributed data sets. As we outlined earlier, we have used Hive, Presto Apache Drill and Spark SQL to query our SDTM model use case and hence do a comparative study of these four solutions in the domain of clinical trials across several standard queries. This benchmark compares the execution time on queries that involve table scans, aggregations and joins. The tools evaluated have a very seamless query transition from ANSI SQL to the big data backend SQL. We next discuss results, which outline the efficiency of executing these queries.

## V. RESULTS

In our results, we outline the hardware and software specifications for our evaluation, the datasets used and the runtime for the various types of queries.

### A. Specifications

#### 1) Hardware Specifications

We used commodity hardware, unlike supercomputers, with three nodes of 2 2.10Ghz Intel Xeon E5-2450, core I7 processor, 32 GB memory, 1 TB 7200 RPM SATA, each currently running CentOS 6.6.

#### 2) Software Specifications

We deployed Cloudera CDH (Cloudera Distribution Including Apache Hadoop), which is the most complete, tested, and popular distribution of Apache Hadoop and related projects. According to Cloudera, [18] CDH is 100% Apache-licensed open source and is the only Hadoop solution to offer unified batch processing, interactive SQL, and interactive search, and role-based access controls till date.

### B. Datasets

We use the clinical trials from CTN (Clinical Trial Network) Dissemination Library created by National Institute on Drug Abuse (NIDA) as input with patients demographic and disposition details for clinical trials.

### C. Queries

Our queries are inspired by the benchmark contained in a comparison of approaches to large-scale analytics. Similar to the queries executed by Pavlo et al [3], we execute the exploratory, join, group by, union queries along with testing the performance with various test conditions in the where clause.

In order to provide an environment for comparing the mentioned analytical tools on the described dataset, we draw workloads and queries from [3].

#### 1) Exploratory Queries

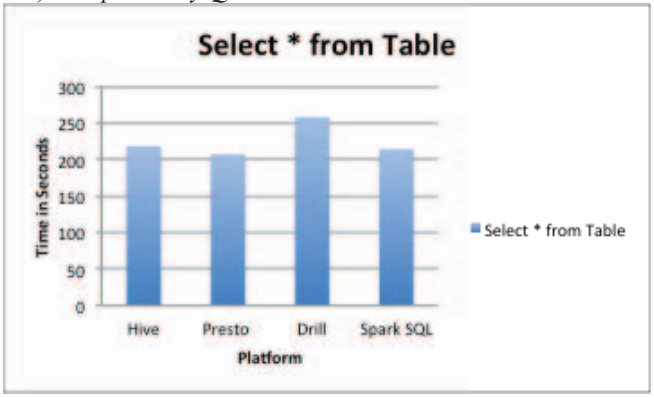


Figure 5: Basic Select Query runtime

In figure 5, we can see that Facebook presto outperforms other big data solutions for the exploratory queries to yield output of 1,62,82,644 rows (2.17 GB).

#### 2) Select Multiple Columns

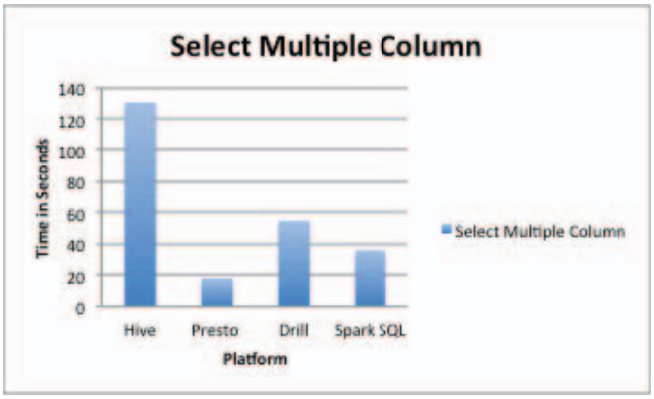


Figure 6: Selecting multiple columns

In figure 6, we get the expected kind of results where in Facebook presto illustrates its power and its effectiveness while querying multiple columns over the dataset of 1,62,82,644 rows.

#### 3) Select Multiple Columns with Where Not condition

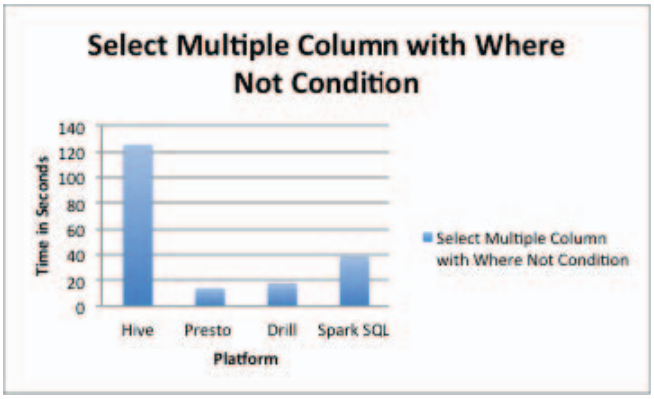


Figure 7: Select with Where not condition

In figure 7, we see the outstanding run time performance of Facebook presto over other big data solutions. Presto follows the push model, which is a traditional implementation of DBMS, processing a SQL query using multiple stages running concurrently. Selecting Multiple Columns is a kind of interactive use case, where Presto can be put to its best use.

#### 4) Join Queries

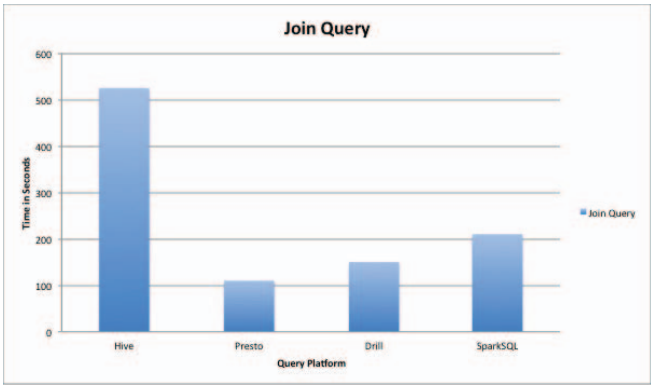

Figure 8: Join Query performance

In figure 8, we see Facebook presto outdoing other big data solutions similar to the queries above. Hive queries get converted into a corresponding mapreduce job. Even when data sets involved are very small (for example, few hundred megabytes), time to execute Hive queries is generally very high. Since SDTM is a STAR based database, Presto following the push model, which is a traditional implementation of DBMS to process a SQL query using multiple stages running concurrently, it is put to its best use of joining large tables with other small tables.

*5) Union All Query*

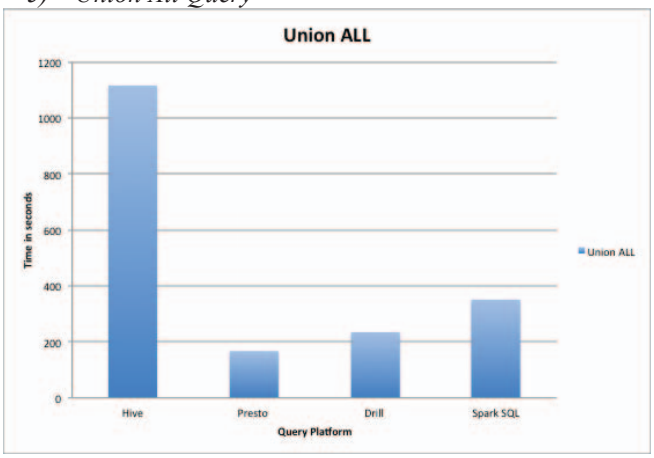

Figure 9: Union All Query performance

In Figure 9, we can see that even in complex Union all queries Facebook presto yields the best runtime over other big data solutions. This is primarily because Presto uses pipe lining of mapreduce commands in order to fetch the query output.

*6) Join Query using different types of surrogate keys*

In Figure 10, we can see that the time taken by the join query using numerical keys (generated by UDFs) as a parameter is almost equivalent to time taken by querying the same table using Java UUID and SHA 256 keys across all four platforms. Thus, we can see that the key generation does not attach any additional overhead.

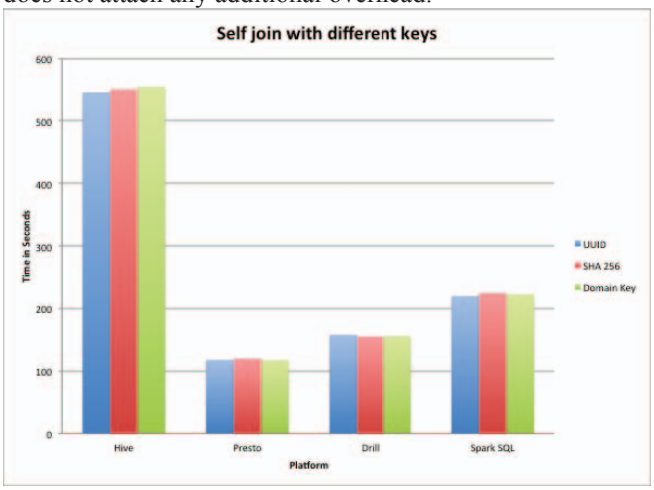

Figure 10: Surrogate key based preformance evaluation

The uniqueness of the generated keys is tested for a data size ranging up to 2.5 GB.
These results suggest that practically unique set of keys can be generated, without collisions using all three proposed methods. Since we have already seen there is no additional overhead involved on join queries, we can use anyone of them to serve our purpose for surrogate keys.

Based on these results and our evaluations we provide some overall other observations and discussion of the results.

## VI. Observations and Discussion

In the table 1, we provide a certain key technology evaluation points against which we evaluate the four tools.

| **Technology/Evaluation Point** | **Hive** | **Presto** | **Drill** | **Spark** |
|---|---|---|---|---|
| Cluster sizes possible | Hundreds of Nodes | Thousands of Nodes | Hundreds of nodes | Thousands of Nodes |
| Data Size Limit | Terabytes | Petabytes | Gigabytes to Petabytes | Petabytes |
| Scalability | High | Very High | High | High |
| ML Algorithms available | No | No | No | Yes |
| Data visualization | No | No | No native support, but works with BI tools with | No |

| | | | | |
|---|---|---|---|---|
| | | | jdbc/odbc | |
| Language Support | Mostly Java API, SQL | C, Java, Node.js, PHP, Python, R, Ruby, SQL | ANSI SQL, Mongo QL, Java AP | Scala, Java, Python, SQL |
| Processing Speed | Fast | 10X faster than MR on disk and 100X faster in memory | Fast | Fast |
| Use case | Batch processing | Real-time Interactive analysis using Query Pipelining | Real-time Interactive analysis | batch processing, streaming, interactive queries, and machine learning. |
| Programming model | MapReduce | MapReduce Pipeline | Queries | MapReduce and DAG |

Table 1: Evaluation Matrix

In general we observe that Hive leverages the mapreduce architecture for data retrieval and transformation. The output of mapreduce is written back to disk. A query in Hive can have many mapreduce jobs, which requires the files to be read from and written to HDFS.

A key advantage of Hive over newer SQL-on-Hadoop engines is robustness: Other engines like Presto require careful optimizations when two large tables are joined. Hive can join tables with billions of rows with ease and should the jobs fail it retries automatically. Presto on the other hand does not use mapreduce. It reads the data from HDFS and has its own proprietary architecture for transforming the data; hence it is much faster compared to Hive.

Our work clearly shows Hive along with Presto outperforming other considered tools. With the presence of machine learning library, Spark SQL surely has an edge for analytics. We can select the tools based on the type of problem at hand.

## VII. Conclusion & Future work

The benchmark proposed in this paper provides an overview of the capabilities of SQL-on-Hadoop platforms. The queries for benchmark have been executed with default tool settings without using optimized configurations. For our experiments, we have addressed a simple comparison between these SQL-on-Hadoop tools based on query execution time and the ease of migrating relational data model to Hadoop based backend considering a potential use case for clinical trial data. The provided benchmark is an implementation of these workloads, which are entirely hosted on an Intel Core i7 processor and can be reproduced from any computer.

In future, we plan to use optimized tool configurations to improve data load process and reduce query-processing time. Hadoop file formats such as Parquet, optimized row columnar (ORC) provide lightweight and fast access to compressed data with columnar layout, hence can significantly boost IO performance. Further, we intend to test the scalability and performance of our system on a bigger cluster. We plan to evaluate other SQL on Hadoop engines such as Cloudera Impala and Hortonworks' Stinger, against large clinical datasets. Using integrated data layer of Hive and NoSQL database such as HBase can also be effective solution for clinical trial analytics.

## ACKNOWLEDGEMENTS

The query outputs reported here result from secondary analyses of data from multiple clinical trials conducted by the National Institute on Drug Abuse (NIDA). NIDA databases are available at http://datashare.nida.nih.gov.

## References

[1] "Clinical Trials." - *NHLBI, NIH*. http://www.nhlbi.nih.gov/studies, Web. 2 Oct. 2015

[2] *Wikipedia*, STDM, https://en.wikipedia.org/wiki/SDTM, Web. 2 Oct. 2015.

[3] Pavlo, Andrew, Erik Paulson, Alexander Rasin, Daniel Abadi, David DeWitt, Samuel Madden, and Michael Stonebraker. "Http://database.cs.brown.edu/sigmod09/benchmarks-sigmod09.pdf." Web. 2 Oct. 2015.

[4] "Big Data: The next Frontier for Innovation, Competition, and Productivity.", http://www.mckinsey.com/insights/business_technology/big_data_the_next_frontier_for_innovation, Web. 2 Oct. 2015.

[5] "General." *Apache Hive TM*. Web. 2 Oct. 2015. https://hive.apache.org/.

[6] "Presto | Distributed SQL Query Engine for Big Data." *Presto | Distributed SQL Query Engine for Big Data*. Web. 2 Oct. 2015.

[7] "Documentation." - *Apache Drill*. Web. 2 Oct. 2015. https://drill.apache.org/docs/.

[8] *Wikipedia*. UUID, Web. 2 Oct. 2015. <https://en.wikipedia.org/wiki/Universally_unique_identifier>.
[9] "Git User's Manual (for Version 1.5.3 or Newer)." *Git User's Manual (for Version 1.5.3 or Newer)*. Web. 2 Oct. 2015. http://schacon.github.io/git/user-manual.html#object-details.
[10] "Clinical Research Continuum: Big Data FOR and FROM Clinical Trials." *Conference Agenda*. Web. 2 Oct. 2015. http://www.clinicalinformaticsworld.com/conference-agenda/ .
[11] "Healthcare & Life Sciences." *Healthcare & Life Sciences*. Web. 2 Oct. 2015. http://www.cloudera.com/content/cloudera/en/solutions/industries/healthcare-life-sciences.html .
[12] Thusoo, Ashish, et al. "Hive: a warehousing solution over a mapreduce framework." Proceedings of the VLDB Endowment 2.2 (2009): 1626-1629.
[13] "Why Presto?" *The Netflix Tech Blog: Using Presto in Our Big Data Platform on AWS*. Web. 2 Oct. 2015. http://techblog.netflix.com/2014/10/using-presto-in-our-big-data-platform.html .
[14] "Hive – The next Generation Data Warehouse." *Hive – The next Generation Data Warehouse*. Web. 2 Oct. 2015. http://blogs.impetus.com/big_data/hadoop_ecosystem/Hive.do.
[15] "Join Optimization in Apache Hive." *Join Optimization in Apache Hive*. Web. 2 Oct. 2015. https://m.facebook.com/notes/facebook-engineering/join-optimization-in-apache-hive/470667928919/?p=10.
[16] Kimball, Ralph. "Newly Emerging Best Practices for Big Data." Web. 2 Oct. 2015. http://www.kimballgroup.com/wp-content/uploads/2012/09/Newly-Emerging-Best-Practices-for-Big-Data1.pdf.
[17] "ClinicalTrials.gov." *Home*. Web. 2 Oct. 2015. https://clinicaltrials.gov/ .
[18] "CDH." *CDH*. Web. 2 Oct. 2015. http://www.cloudera.com/content/cloudera/en/products-and-services/cdh.html .
[19] "Presto versus Hive | Treasure Data Blog." *Treasure Data Blog RSS*. 20 Mar. 2015. Web. 2 Oct. 2015. http://blog.treasuredata.com/blog/2015/03/20/presto-versus-hive/ .
[20] "Big Data: How Do Your Data Grow?" Web. 2 Oct. 2015. http://www.nature.com/nature/journal/v455/n7209/full/455028a.html .
[21] Berman, Jules J. *Principles of Big Data Preparing, Sharing, and Analyzing Complex Information*. Morgan Kaufaman, 2013.
[22] Zhan, Jianfeng. *Big Data Benchmarks, Performance Optimization, and Emerging Hardware: 4th and 5th Workshops, BPOE 2014, Salt Lake City, USA, March 1, 2014 and Hangzhou, China, September 5, 2014 : Revised Selected Papers*.
[23] Balkić, Zoran, Damir Sostaric, and Goran Horvat. "GeoHash and UUID Identifier for Multi Agent Systems." *ResearchGate*. Web. 2 Oct. 2015. http://www.researchgate.net/publication/232285663_GeoHash_and_UUID_identifier_for_Multi_agent_systems .
[24] Hunt, Tim D. (2010) Implementing a UUID primary key in a distributed email client application. In: Proceedings of the 1st Annual Conference of Computing and Information Technology Education and Research in New Zealand (CITRENZ): Incorporating the 23rd Annual Conference of the National Advisory Committee on Computing Qualifications. National Advisory Committee on Computing Qualifications (NACCQ), Hamilton, New Zealand, pp. 71-78.
[25] Iordanov, Borislav. "HyperGraphDB: A Generalized Graph Database." Web-age Information Management WAIM 2010 International Workshops: IWGD 2010, XMLDM 2010, WCMT 2010, Jiuzhaigou Valley, China, July 15-17, 2010 : Revised Selected Papers. Berlin: Springer, 2010.
[26] Devi, P.Sarada, V.Visweswara Rao, and K. Raghavender. "Emerging Technology Big Data-Hadoop over Datawarehousing ETL." Web. 2 Oct. 2015. http://iraj.in/up_proc/pdf/103-141145144330-34.pdf .
[27] Floratou, Avrilia, Umar Farooq Minhas, and Fatma Ozcan. "SQL-on-Hadoop: Full Circle Back to Shared-Nothing Database Architectures." Web. 2 Oct. 2015. http://www.vldb.org/pvldb/vol7/p1295-floratou.pdf .
[28] Kornacker, Marcel, Alexander Behm, Victor Bittorf, Taras Bobrovytsky, Casey Ching, Alan Choi, Justin Erickson, Martin Grund, Daniel Hecht, Matthew Jacobs, Ishaan Joshi, Lenni Kuff, Dileep Kumar, Alex Leblang, Nong Li, Ippokratis Pandis, Henry Robinson, David Rorke, Silvius Rus, and John Russell. "Impala: A Modern, Open-Source SQL Engine for Hadoop."
7th Biennial Conference on Innovative Data Systems Research, CIDR 2015.