Towards a Big Data Analytics Platform with Hadoop/MapReduce Framework using Simulated Patient Data of a Hospital System

By

Dillon Chrimes
BSc, Health Information Science, University of Victoria, 2012
PhD, Forest Ecology & Management, Swedish University of Agricultural Sciences, 2004
MSc, Silviculture, Swedish University of Agricultural Sciences, 2001
BSc, Sustainable Forest Management, University of Alberta, 1997

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science in the School of Health Information Science

Supervisory Committee

Towards a Big Data Analytics Platform with Hadoop/MapReduce Framework using Simulated Patient Data of a Hospital System

By

Dillon Chrimes

BSc, Health Information Science, University of Victoria, 2012
PhD, Forest Ecology & Management, Swedish University of Agricultural Sciences, 2004
MSc, Silviculture, Swedish University of Agricultural Sciences, 2001
BSc, Sustainable Forest Management, University of Alberta, 1997

Supervisory Committee

Dr. Alex (Mu-Hsing) Kuo, School of Health Information Science, Department of Human and Social Development, University of Victoria
Supervisor

Dr. Andre Kushniruk, School of Health Information Science, Department of Human and Social Development, University of Victoria
Departmental Member

Abstract

Background: Big data analytics (BDA) is important to reduce healthcare costs. However, there are many challenges. The study objective was high performance establishment of interactive BDA platform of hospital system.

Methods: A Hadoop/MapReduce framework formed the BDA platform with HBase (NoSQL database) using hospital-specific metadata and file ingestion. Query performance tested with Apache tools in Hadoop's ecosystem.

Results: At optimized iteration, Hadoop distributed file system (HDFS) ingestion required three seconds but HBase required four to twelve hours to complete the Reducer of MapReduce. HBase bulkloads took a week for one billion (10TB) and over two months for three billion (30TB). Simple and complex query results showed about two seconds for one and three billion, respectively.

Interpretations: BDA platform of HBase distributed by Hadoop successfully under high performance at large volumes representing the Province's entire data. Inconsistencies of MapReduce limited operational efficiencies. Importance of the Hadoop/MapReduce on representation of health informatics is further discussed.

Table of Contents

List of Tables

List of Figures

Acknowledgments

I would like to acknowledge family and friends for their support. I would also like to acknowledge VIHA's Research Capacity and Ethical Support. The VIHA staff experts are thanked for their continued support of this project. Special thanks to continued success in research with Professor Hideo Sakai (University of Tokyo) and Dr. Mika Yoshida (University of Tsukuba) from Japan, and their research international excursions to Victoria BC, Canada. Also, Sierra System Group Inc., especially Karl, Rayo, and Sarah are thanked for support and consultation with BI teams and important financial contribution in this project's early stages. Dr. Alex Kuo is thanked for his ongoing support and research discussion, as well as Dr. Andre Kushniruk. Thanks go to Dr. Elizabeth Borycki for career advice and ongoing discussions. WestGrid Administrators, especially Dr. Belaid Moa, thanks for your ongoing technical ideas, implementations, and enhanced administrative support. Hamid Zamani is thanked for coffee breaks and operational support as research assistant. Gerry Bliss and Sharif are both thanked for initial discussions for broader research of security/privacy on Hadoop ecosystem. Dr. Roudsari is thanked for positive discussions on GIS, data visualizations and programming. Dr. Shabestari is thanked for initial discussion and planning of BI tools and systems. Thanks to Glen Vajcner for the help during the Calgary floods of 2013 and quick chats on thesis work, as well as Paul Payne.

Dedication

I'd like to dedicate the start of my thesis firstly to my family and friends, whose friendship, discussions and encouragement is true to my success. Mostly all is dedicated to my dog, Flying Limorick Rowan for many weekend walks and important breaks in between the thesis writing. And I would like to dedicate the latter part of the thesis to a dear friend, Valentina Contreras Mas, who I've missed continually.

1. Research Background/Motivation

1.1 Challenges in Health Informatics, Data Mining, and Big Data

As a cross-sectional discipline health informatics forms an important basis of modern medicine and healthcare (Nelson & Staggers, 2014). Gantz and Reinsel (2012) predicted in their 'The Digital Universe' study that the digital data created and consumed per year will reach 40,000 Exabyte by 2020, from which a third will promise value to organizations if processed using big data technologies. However, the increase in digital data and fluid nature of information-processing methods and innovative big data technologies has not caused an increase of implementations in healthcare. There are very few if any of the 125 countries surveyed by the World Health Organization with any Big Data strategy for universal healthcare coverage of their eHealth profiles (WHO, 2015). Tsumoto, Hirano, Abe, and Tsumoto (2011) did implement a seamless interactivity with automated data capture and storage; nonetheless, like many others, it was on a very small scale. Thus, the challenge of data mining medical histories of patient data and representative health informatics in real time at large volumes remains a very daunting task (Hughes, 2011; Hoyt, Linnville, Chung, Hutfless, & Rice, 2013; Fasano, 2013). Wang, Li, and Perrizo (2014) describe the extraction of useful knowledge from Big Data in terms of a processing pipeline that transfers, stores, and analyzes data for whole systems. According to Kuo, Sahama, Kushniruk, Borycki, and Grunwell (2014), the process of achieving full data acquisition and utilization for a platform to healthcare applications involves five distinct configuration stages of a Big Data Analytics (BDA) platform, each of which presents five specific challenges (that formed this study's objectives):

i. Data aggregation

Currently, aggregating large quantities of data usually involves copy/transfer the data to a dedicated storage drive (cloud services still remain an outlier in many sectors although virtual machines is the norm and increasing). The dedicated storage that drives the exchange or migration of data between groups and databases can be very time consuming to coordinate with ongoing maintenance, especially with big data that often involves a secondary productive system to share the production resources (Maier, 2013). Network resources with low latency and high bandwidth are desirable to transfer data for *ftp* or via secure s*ftp*. However, transferring vast amounts of data into or out of a data repository poses a significant networking and bandwidth challenge (e.g., Ahuja & Moore, 2013). With big data technologies, extremely large amounts of data can be replicated from the sources and generated iteratively across domain instances as a distributed file system. Hadoop's distributed file system (HDFS) replicates its HFiles into blocks and can store them in a rack-aware server hardware/software technology nature across several DataNodes (Grover, Malaska, Seidman, & Shapira, 2014; Lith & Mattson, 2010; White, 2015).

ii. Data maintenance

Since Big Data are very large collections of datasets, it is very difficult to maintain the data intact at each sequence of queries and reporting. Even with using traditional data-management systems, such as relational databases using HL7 and other healthcare data standards (Umer, Afzal, Hussain, Latif, & Ahmad, 2016), maintaining several clinical tables, for example for patient registries, is a major daily operational task. There are constant updates in clinical

reporting to hospitals and healthcare organizations to maintain accurate representation of real patient data, metadata, and data profiles; otherwise, the analytics is rendered useless (Kuo et al., 2014; Wang, et al., 2014). Moreover, delayed time and increased money required to implement/maintain can prohibit small organizations from managing the data over the long run for clinical reporting, especially over a big data platform for healthcare that requires solid data integrity with high accuracy.

iii. Data integration

Data integration involves standardized protocol and procedures in maintenance of metadata from several parts of the system by combining and transforming data into an appropriate format for analysis in a data warehouse. Since Big Data in healthcare is about systems distributed with all data, structured to unstructured, combined to preserve and manage effectively the variety and heterogeneity of the data over time, it is extremely challenging and most times not possible (Dai, Gao, Guo, Xiao, & Zhang, 2012; Marin-Sanchez & Verspoor, 2014; Seo, Kim, & Choi, 2016). A HDFS framework can encompass large-scale multimedia data storage and processing in a BDA platform for healthcare. The integrated patient data needs to be established in the database accurately with HDFS in order to avoid errors and to maintain data integrity and high performance (Maier, 2013; Lai, Chen, Wu, & Obaidat, 2014). Currently, even the sole integration of structured Electronic Health Record (EHR) data only is a major challenge for any operating hospital system (Kuo, Kushniruk, & Borycki, 2010; 2011).

iv. Data analysis

Complexity of the analysis involves analytic algorithms to not only maintain but to enhance performance. Big data has been deemed a tool to find unknown or new trends that can reduce costs in healthcare (Canada Health Infoway, 2013; Kayyali, Knott, & Van Kuiken, 2013). It is important for the analysis not only analyze the data similar to conventional ways but also that much more to larger vast datasets at a performance better than what the current operating system can accomplish. Since computing time increases dramatically even with small increases in data volume (e.g., Kuo et al., 2014), hardware and software utilization to perform the analysis becomes major issues or barriers if not working properly and, therefore, could derail the BDA utilization all together. For example, in the case of the Bayesian Network, a popular algorithm for modeling knowledge in computational biology and bioinformatics, the computing time required to find the best network increases exponentially as the number of records rises (Schadt, Lindermann, Sorenson, Lee, & Nolan, 2010). There are also ontological approaches to analytics using big data that use advanced algorithms similar to Bayesian Networks (Kuiler, 2014). Similarly, scale of the data is important to big data architects (e.g., Maier, 2013). Even for simple analysis, it can take several days, even months, to obtain a result when over very large datasets (e.g., on the Zettabyte scale).

Many state that parallelization of computing model is required by many Hadoop technologies (White, 2015; Yu, Kao, & Lee, 2016). For some computationally intense problems, Hadoop/MapReduce programmable framework can be efficiently parallelized so that tasks can be distributed among many hundreds or even thousands of computers (Marozzo, Talia, & Trunfio, 2012; Mohammed, Far, & Naugler, 2014; Mudunuri et al., 2013). However, studies do indicate that Hadoop cannot be parallelized and unable to harness the power of the massive parallel-processing tools (e.g., Sakr & Elgammal, 2016; Wang et al., 2014).

v. Pattern interpretation of value of application for healthcare

Data validation of clinical reports is important. Many clinical reporters, managers and executives instinctively believe that bigger data that can show trending on dashboards will always provide better information for decision-making (e.g., Grossglauser & Saner, 2014). Unfortunately, agile data science cannot protect us from inaccuracies, missing information, faulty assumptions, positive negatives, etc... Many analysts and reporters can often be fooled into thinking everyone can understand correlations that emerge from analysis when their true significance (or lack thereof) is hidden in the nuances of the data, its quality, and its structure. In fact, some studies show that the trustworthiness of Big Data seems to be unsurpassed by current reporting (Mittal, 2013). Knowledge representation is an absolute must for any data mining and BDA platforms (e.g., Li, Park, Ishag, Batbaatar, & Kyu, 2016). Furthermore, a BDA platform is of little value if decision-makers do not understand the patterns it discovers and cannot use the trends to reduce costs or improve processes. Also, given that different relationships in data can be derived when data are combined, the connectivist approach (historically considered the go-to theory supporting learning in the digital age) takes ideas from brain models and neural networks in learning from technologies and how massive amounts of data can contribute and enhance this form of learning (Siemens, 2004). Unfortunately, given the complex nature of data analytics in healthcare, it is challenging to represent and interpret results in a form that is comprehensible to non-experts.

Legality and ethics is a major issue to contend with in utilization of large datasets of patient data in healthcare (cf. Johnson & Willey, 2011). This is the case not only in healthcare but also even in most legal professions that have had big data technologies applied to services– Judges Analytics by Ravel Law (for example)– *"…lets lawyers search through every decision made by particular judges to find those most likely to be sympathetic to their arguments"* (Marr, 2016). The protection of patient confidentiality in the era of Big Data is technologically possible but challenging (Schadt, 2012). In healthcare, security, confidentiality, and privacy of patient data are mandated by legislation and regulations. For example, the Health Insurance Portability and Accountability Act (HIPAA), as well as Freedom of Information and Protection of Privacy (FIPPA) Act requires the removal of 18 types of identifiers, including any residual information that could identify individual patients (e.g., Moselle, 2015). These privacy mandates are a major barrier for any BDA implementation and utilization. With large datasets, it is all too easy to unveil significant patient information that is a breach of confidentiality and against the rules of public disclosure. Privacy concerns can be addressed using new technologies, such as key-value (KV) storage services, but advanced configuration and technical knowledge is needed during implementation and afterwards for ongoing operational maintenance. For example, Pattuk, Kantarcioglu, Khadilkar, Ulusoy, and Mehrotra (2013) proposed a framework for securing Big Data management involving an HBase database – called Big Secret – securely outsources and processes encrypted data over public KV stores.

Data privacy in healthcare involves restricted access to patient data but there are often challenging situations when using hospital systems and attempting to find new trends in the data. For instance, on one hand there are workarounds to access patient data in critical situation like sharing passwords that goes against HIPAA and FIPPA Acts (Koppel, Smith, Blythe, & Kothari, 2015). There are strict rules and governance on hospital systems with advanced protection of privacy of patient data based on HIPAA (Canada Health Infoway and Health Information Privacy Group, 2012; Kumar, Henseler, & Haukass, 2009) that must take into consideration

when implementing a BDA platform. It's processing and storage methods must adhere to data privacy at a high level and also the accessibility of the data for public disclosure (Erdmann, 2013; Spiekermann & Cranor, 2009; Win, Susilo, & Mu, 2006). One method of ensuring that patient data privacy/security is to use indexes generated from HBase, which can securely encrypt KV stores (Chawla & Davis, 2013; Chen et al., 2015; Xu et al., 2016), and HBase can further encrypt with integration with Hive (Hive HBase, (2016).

In a hospital system, such as for the Vancouver Island Health Authority (VIHA), the capacity to record patient data efficiently during the processes of admission, discharge, and transfer (ADT) is crucial to timely patient care and the quality of patient-care deliverables. Sometimes the ADT system is referred to as the source of truth for reporting of the operations of the hospital from inpatient to outpatient and discharged patients. Among these deliverables are reports of clinical events, diagnoses, and patient encounters linked to diagnoses and treatments. Additionally, in most Canadian hospitals, discharge records are subject to data standards set by Canadian Institute of Health Information (CIHI) and entered into Canada's national Discharge Abstract Database (DAD). Moreover, ADT reporting is generally conducted through manual data entry to a patient's chart and then it is combined with EHR (which could also comprise auto-populate data) that might consist of other hospital data in reports to provincial and federal health departments (Ross, Wei, & Ohno-Machado, 2014). These two reporting systems, i.e., ADT and DAD, account for the majority of patient data in hospitals, but they are seldom aggregated and integrated as a whole because of their complexity and large volume. A suitable BDA platform for a hospital should allow for the integration of ADT and DAD records and to query that combination to find unknown trends at extreme volumes of the entire system.

The methodology of data mining differs from that of traditional data analysis and retrieval. In traditional methodology, records are returned via a structured query; in knowledge discovery (e.g., Fayyad, Piatestky-Shapiro, & Smith, 1996), what is retrieved in the database is implicit rather than explicitly known (Bellazzi et al., 2011). Thus, data mining finds patterns and relationships by building models that are predictive or prescriptive but not descriptive. Diagnostic analytics is, however, entrenched in the data-mining methodology of health informatics (Podolak, 2013). Chute (2005) points out that health informatics are biased towards the classification of data as a form of analytics, largely, in the Case in Canada, because the data standards of the DAD are set by CIHI for clinical reporting. Unfortunately, in this study, only the structured and classified data was formulated in a simulation to form a database and query. Nevertheless, proprietary hospital systems for ADT also have certain data standards that are partly determined by the physical movement of patients through the hospital rather than the recording of diagnoses and interventions. Therefore, as a starting point, the structured and standardized data can be easily benchmarked and with simple performance checks. There are also legal considerations to contend with when producing results from data mining of overall systems and this includes the threat of lawsuits. All restrictions limit the data that gets recorded, especially on discharging a patient a physician is legally required only to record health outcomes rather than the details of interventions. For these and other reasons, health informatics has tended to focus on the structure of databases rather than the performance of analytics at extreme volumes.

The conceptual framework for a BDA project in healthcare is similar to that of a traditional health informatics or analytics project. That is, its essence and functionality is not totally different from that of conventional systems. The key difference lies in data-processing methodology. In terms of the mining metaphor, data represent the gold over the rainbow while analytics systems represent the leprechaun that found the treasure or the actually mechanical minting of the metals to access it. Moreover, healthcare analytics is defined as a set of computer-based methods, processes, and workflows for transforming raw health data into meaningful insights, new discoveries, and knowledge that can inform more effective decision-making (Sakr & Elgammal, 2016). Data mining in healthcare has traditionally been linked to knowledge management, reflecting a managerial approach to the discovery, collection, analysis, sharing, and use of knowledge (Chen, Fuller, Friedman, & Hersh, 2005; Li et al., 2016). Thus, the DAD and ADT are designed to enable hospitals and health authorities to apply knowledge derived from data recording patient numbers, health outcomes, length of stay (LOS), and so forth, to the evaluation and improvement of hospital and healthcare system performance. Furthermore, because the relational databases of hospitals are becoming more robust, it is possible to add columns and replicate data in a distributed filing system with many (potentially cloud-based) nodes and with parallel computing capabilities. The utility of this approach is that columns can be combined (e.g., columns from the DAD and ADT). And such a combination can mimic data in the hospital system in conjunction with other clinical applications. Through replication and ingestion, the columns can form one large file that can then be queried (and columns can be added and removed or updated), which was the goal of what this study set out to do.

1.2 Implementation of Big Data Analytics (BDA) Platform in Healthcare

Healthcare authorities and hospital systems need BDA platforms to manage and derive value from existing and future datasets and databases. Under the umbrella of the hospital system with its end users, the BDA platform should harness the technical power and advanced programming of accessible front-end tools to analyze large quantities of back-end data in an interactive manner, while enriching the user experience with data visualizations. All this must be accomplished at moderate expense for a successful platform to be used.

To test the application of BDA in healthcare, a study conducted research in Victoria, British Columbia, using simulated patient data. Building on previous works (Chrimes, Kuo, Moa, & Hu, 2016a; Chrimes, Moa, Zamani, & Kuo, 2016b; Kuo, Chrimes, Moa, & Hu, 2015), the broader study established a Healthcare BDA (HBDA) platform at the University of Victoria (UVic) in association with WestGrid (University of Victoria, 2013; 2016) and Vancouver Island Health Authority (VIHA). The HBDA framework and platform emulated patient data from two separate systems: ADT (hospital-generated admission, discharge, and transfer records) and DAD (CIHI's (2012) National Discharge Abstract Database), represented in the main hospital system and its data warehouse stored at VIHA. The data was constructed and cross-referenced with data metadata of current and potential clinical reporting at VIHA. It was then validated by VIHA for use in testing the HBDA platform and its performance in several types of queries of patient data. This overall simulation was a proof-of-concept implementation of queries of patient data at large volumes over a variety of BDA configurations, scripts and performances. In this study, it is important that data was emulated only from structured textual patient data that was already defined by metadata because its data profiles had to be representative of the real data of the hospital system.

While the potential value of BDA, as applied to healthcare data, has been widely discussed, for example by Canada Health Infoway (2013) stated that big data technologies can reduce healthcare costs, real-world tests have rarely been performed, especially for entire hospital systems. As an improvement of over traditional methods of performing relational database queries, the BDA platform should offer healthcare practitioners and providers the necessary analytical tools. And these tools need to manage and analyze large quantities of data interactively, as well as enrich the user experience with visualization and collaboration capabilities using web-based applications. In this project, it was, therefore, proposed to establish a framework to implement BDA to process and analyze patient data to simulate a representation of very large data analytics of an entire health authority's hospital system and its archived history within the Province. No such methodology is currently utilized in an acute-care setting at a regional health authority or provincially.

Currently, the data warehouse at VIHA has over 1000 relational tables of its kernel ADT system that encompass a total of approximately 1-10 billion records archived over a period of ~50 years (personal communication). Health data are continuously generated and added to the warehouse at a rate that grows exponentially over the past decade. The clinical data warehouse basically comprises ADT and DAD records. ADT represents the entire VIHA hospital system and data profiles used for reporting, while DAD contains CIHI diagnostic codes and discharge-abstract metadata. Currently, VIHA uses numerous manual and abstracting processes to combine patient data over the main relational databases of the hospital system. However, neither business intelligence (BI) tools nor data warehouse techniques are currently applied to both databases (ADT and DAD) at the same time and over its entire volume of data warehouse.

The DAD is comprised of mostly diagnostic data with some ADT-like data derived from other sources, e.g., EHR, in the system and at the hospital. The user must be entered manually to abstract clinical reporting into the DAD, because neither diagnoses nor interventions nor provider interactions are captured by the ADT system. Unit clerks, nurses, physicians, and other health professionals enter data in EHRs for inpatient visits, as well as ADT data. The kind of questions BDA might help to answer is that clinical reporters and quality and safety advisors suspect that frequent movement of patients within the hospital can worsen outcomes, e.g., increase sepsis occurrences. This may be especially true of those (e.g., the elderly) who are prone to environmental stress. Moreover, misdiagnosis can result; for example, from a physician diagnosing the onset of an infection, a disease, or a mental illness during an agitated state with patient movement, resulting in unnecessary laboratory tests or medications. To simulate the benefit of combining ADT with DAD, this study can employ data mining of ADT and DAD records to discover the impact of frequent bed movements on patient outcomes.

The implementation allowed the construction of three billion records for testing the HBDA platform, which was based on existing data profiles and metadata in the hospital system. The simulation was intended to demonstrate significant improvements in query times, the usefulness of the interfaced applications, and the overall usefulness of the platform that shows it can leverage existing ADT-DAD workflows and SQL codes. In this approach to implementation challenges, it was conceptualized the HBDA as a pipelined data-processing framework (Chrimes et al., 2016a; Chrimes et al., 2016b; Kuo et al., 2015) that are visualized in Figures 1 and 2).

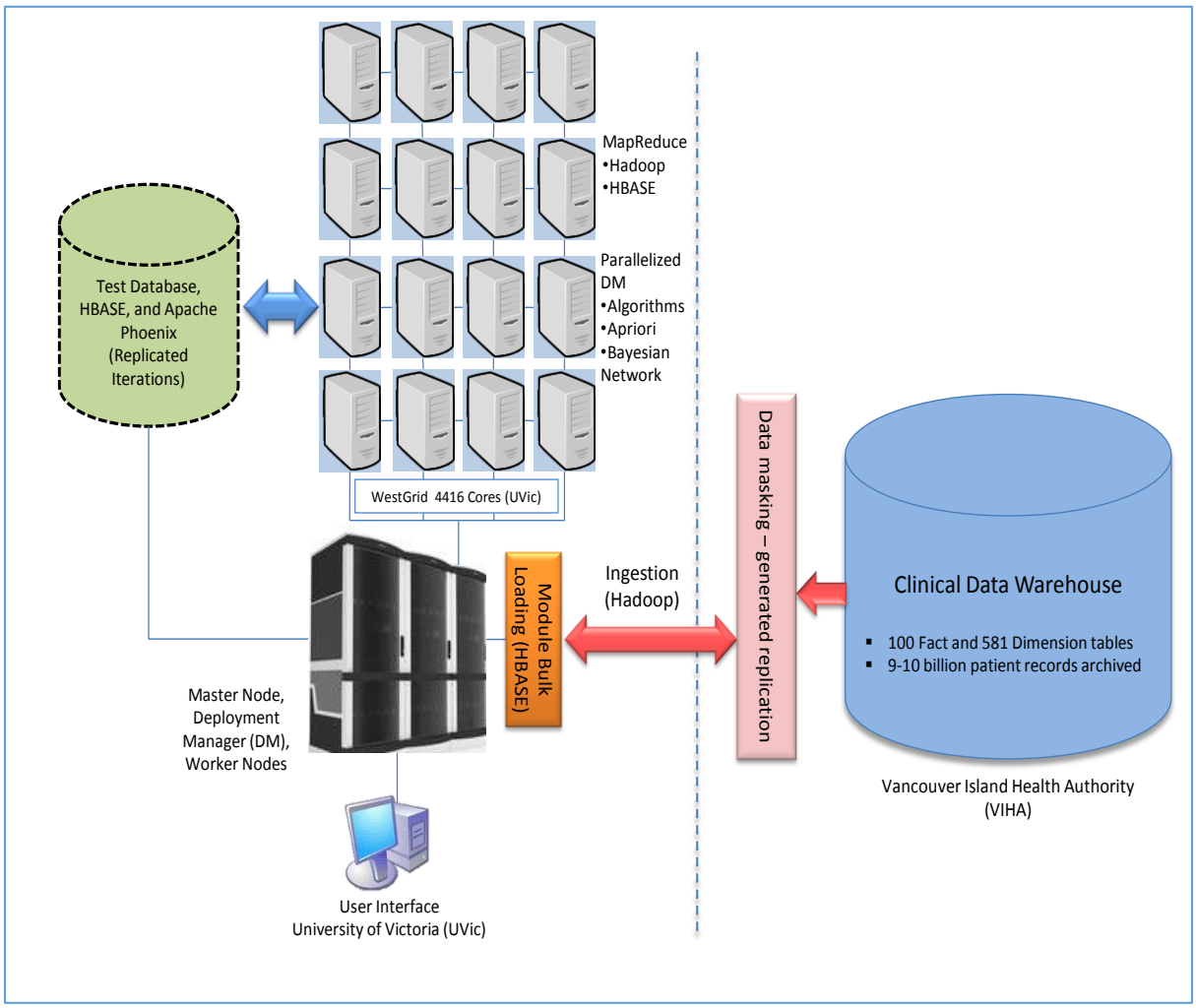

Figure 1. The proposed Health Big Data Analytics (HBDA) platform framework with Vancouver Island Health Authority with masked or replicated Hadoop distributed filing system (HDFS) to form NoSQL HBase database via MapReduce iterations with big data tools interacting with the NoSQL and HDFS under parallelized deployment manager (DM) at WestGrid, UVic.

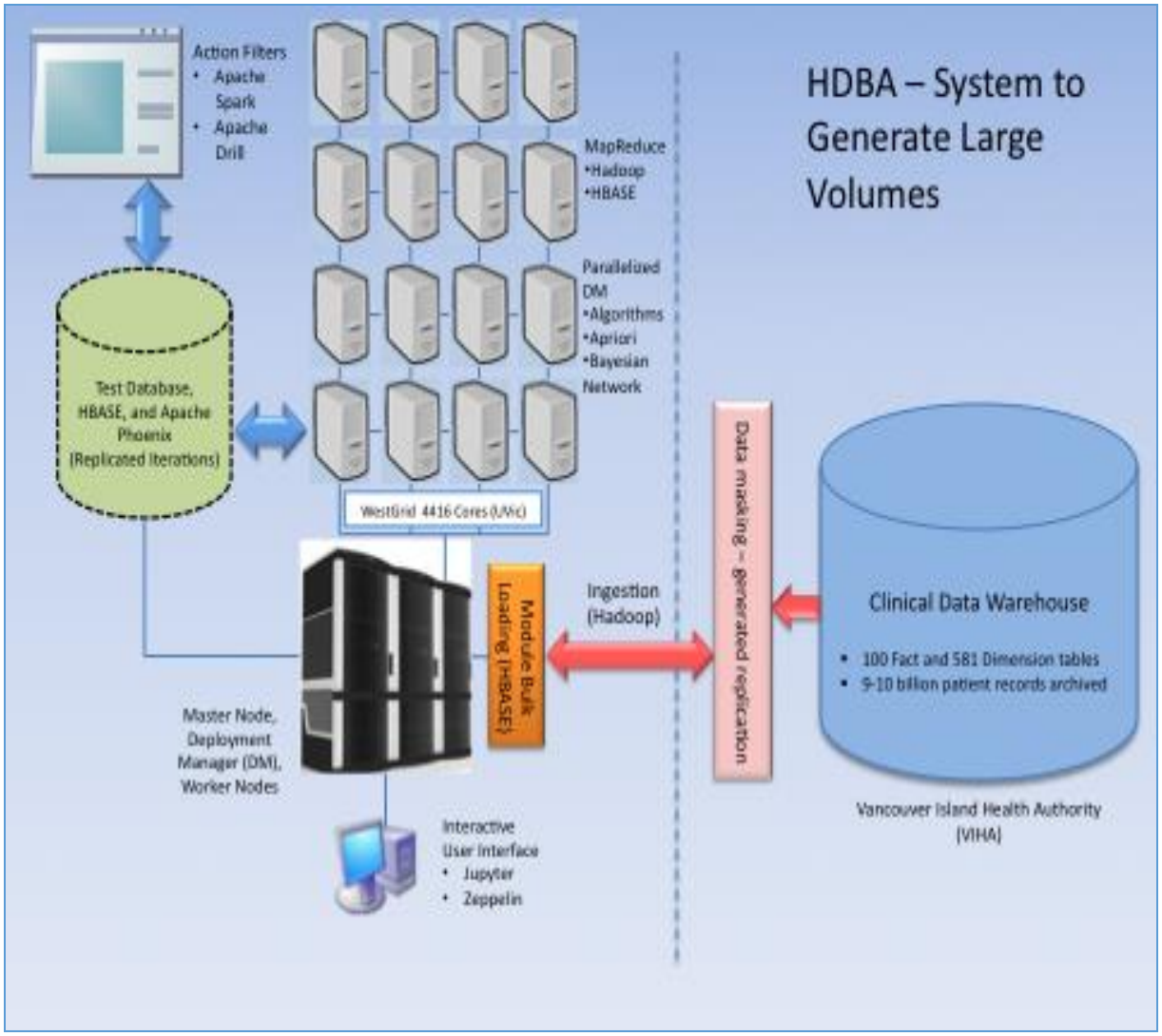

Figure 2. The proposed Health Big Data Analytics (HBDA) platform framework with Vancouver Island Health Authority with masked or replicated Hadoop distributed filing system (HDFS) to form NoSQL HBase database with Apache Spark and Apache Drill interfaces via MapReduce iterations with big data tools interacting with the NoSQL and HDFS under parallelized deployment manager (DM) at WestGrid, UVic.

There were also interviewed stakeholders at VIHA to identify the important metadata of inpatient profiles (Figure 3). The workflow processes used in generating reports and the applications used for querying results were also discussed in these stakeholder interviews.

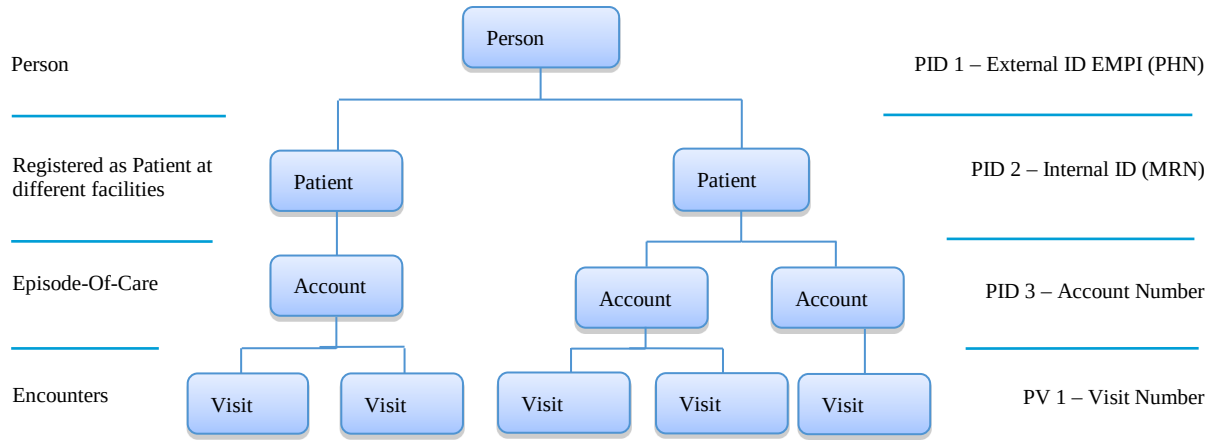

Figure 3. The ADT system is based on episode-of-care as a patient centric data model. PID is Patient Identification of Personal Health Number (PHN) in the enterprise master patient index (EMPI), Medical Record Number (MRN) at different facilities, and PV is Patient Visit for each encounter for each episode-of-care.

The simulated patient data was constructed from metadata and data profiles in VIHA's data warehouse, corroborated by VIHA experts, and configured in 90 selected columns of combined DAD and ADT data. Thus, this study's framework was able to test the BDA platform and its performance on emulated patient data as an accurate representation of the data, workflow to combining the ADT and DAD, and the actual queries and results expected.

The established BDA platform was used to test query performance, using actions and filters that corresponded to VIHA's current reporting practices. The first step was to emulate the metadata and data profiles of VIHA's ADT model, which is the proprietary Cerner hospital system. The metadata derived for patient encounters was combined with VIHA's hospital reporting to the DAD. The data aggregation represented both the source system of patient encounters and its construct, which represented patient data collected during each encounter (for various encounter types), from admission to discharge. VIHA's Cerner system uses hundreds of tables to represent ADT, all of which are included in a relational database with primary and foreign keys. These keys are important for data integrity in that their qualifiers link clinical events to individual patients. Therefore, the patient data construct had to use constraints in emulating the data to form a plausible NoSQL database of a hospital system.

The objective was to establish an interactive dynamic framework with front-end and interfaced applications (i.e., Apache Phoenix, Apache Spark, and Apache Drill) linked to the Hadoop HDFS and back-end NoSQL database of HBase to form a platform with big data technologies to analyze very large volumes. By establishing a platform, challenges of not only implementing but amalgamated to healthcare scenarios of Big Data can be applied and overcome. Together, the framework and the applications over the platform would allow users to visualize, query, and interpret the data. The overall purpose is to make Big Data capabilities accessible to stakeholders, including physicians, VIHA administrators, and other healthcare practitioners (Figure 4).

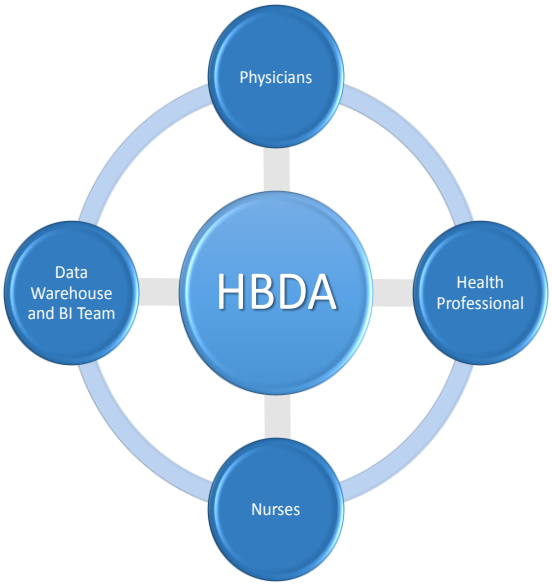

Figure 4. Main stakeholder groups (Physicians, Nurses, Health Professionals, and Data Warehouse and Business Intelligence (BI) team), at VIHA involved in clinical reporting of patient data with Admission, Discharge, and Transfer (ADT) and Discharge Abstract Database (DAD). This includes physicians, nurses, health professionals (epidemiologists and clinical reporting), and data warehouse and business intelligence (BI).

There were a few hypotheses based on the five challenges to implement and test a platform of BDA in healthcare. The NoSQL database created using hospital and patient data in differentiated fields would accurately simulate the patient data while emulating all queries, with provisions for both simple and complex queries. Simple queries would show faster query times compared to complex and this would increase as the volume increased. Another hypothesis was that high performance could be achieved by using a number of nodes optimized at the core CPU capacity. Furthermore, there should be differences in the performance times of ingestion and queries of the big data technologies tested because they have fundamentally different configurations, which is outlined in detail in the Material and Methods 3.6 and 3.7 sections. Lastly, patient data could be established as secure based on HBase/Hadoop architecture and heavily relying on WestGrid's High Performance Computing via sponsored log in and access to run batch process with the Big Data Hadoop (open source) software installed.

2. Literature Review

2.1 Big Data Definition and Application

Large datasets have been in existence for hundreds of years, beginning in the Renaissance Era when researchers began to archive measurements, pictures, and documents to discover fundamental truths of outcomes in nature (Coenen, 2004; Haux, 2010; Hoyt et al., 2013; Ogilive, 2006), and thereafter including the surmise of Darwin's natural selection theory in 1859 (Hunter, Altshuler, & Rader, 2008). In the field of health informatics, John Snow's 1850s mapping and investigation of the source of cholera in London demonstrated a method of data mining and visual analytics applicable to large data sets (Chen, Chiang, & Storey, 2012; Hempel, 2007; Kovalerchuk & Schwig, 2004). Today, we are beyond small studies and in a large digital shadow growing rapidly for more than Exabytes (e.g., Gantz & Reinsel, 2012). In healthcare, patient data is compiled in hospital datasets in large data warehouses. The data are record case-by-case instances of patient-physician interactions with defined outcomes, resulting in large volumes of information that is now sometimes referred to as Digital Healthcare.

The term "Big Data" was introduced in 2000 by Francis Diebold, an economist at the University of Pennsylvania, and became popular when IBM and Oracle adopted it in 2010 (i.e., Nelson & Staggers, 2014, page 24). Big Data refers not only to the sheer scale and breadth of large datasets but also to their increasing complexity. A widely used mnemonic to describe the complexity of Big Data is the "three V's": volume, variety, and velocity (e.g., Canada Health Infoway, 2013). Conventional methods of data processing become problematic when confronted with any combination of database size (volume), frequency of update (velocity), or diversity (variety) (Chen, Mao, & Liu, 2014; Klein, Tran-Gia, & Hartmann, 2013; Mohammed et al., 2014; Taylor, 2010). To this list, some authorities add a fourth V, veracity, which refers to the quality of the data. While not a defining characteristic of Big Data *per se*, inconsistencies in veracity can affect the accuracy of analysis and essentially the performance of the platform. As data sets continue to become larger and more complex, the computational infrastructure required to process them also increases; therefore, big data technology have increasingly developed to improve performance over computing systems.

The growing need for efficient data mining and analysis in many fields has stimulated the development of advanced distributed algorithms and BDA (Cumbley & Church, 2013; Ferguson, 2012; Langkafel, 2016; Quin & Li, 2013). There is some push back from ethical issues involving patient-provider anaesthesia treatments, as outlined in an ethical perspective by Docherty (2013); however, de-identified data at extreme volumes remains an advantage for BDA to apply to healthcare. BDA consists of the application of advanced methodologies to achieve the level of processing and predictive power needed to mine large datasets and discover and track emerging patterns and trends (Fisher, DeLine, & Czerwinski, 2012; Mehdipour, Javadi, & Mahanti, 2016; Tien, 2013). It is particularly important in the field of healthcare, which has a larger array of data standards and structures than most other industries (Nelson & Staggers, 2014) combined with a strong demand for knowledge derived from clinical text stored in hospital information systems (Alder-Milstein, & Jha, 2013).

Broadly, BDA can be viewed as a set of mechanisms and techniques, realized in software, to extract "hidden" information from very large or complex data sets (Cios, 2001; Miller & Han,

2009; Wigan & Clarke, 2013). The word *hidden* or unknown is important, but there are differing views as to what hidden data are and how they influence methodology and analysis. A "Not Only Workload Generator" or noWOG engine was designed to test database performance of big data workloads revealed that querying their simulated databases by means of Structured Query Language (SQL) or SQL-styled techniques reveals hidden information (Ameri, Schlitter, Meyer, & Streit, 2016); however, SQL, although somewhat sophisticated, is not always viewed as an appropriate Big Data tool for use in many fields like in health informatics (Bellazzi et al., 2011). Similarly, algorithms have been mentioned in several studies as defining data mining for BDA by forensically finding hidden information or combining hidden, unknown, and previously established information in Big Data. Clearly, the increasing volume of Big Data and its increasing velocity (often as much as 100 times faster than traditional databases, Baro, Degoul, Beuscart, & Chazard, 2015) pose some significant challenges to analytics, including the need to perform broader queries on larger data sets, accommodate machine learning, and employ more advanced algorithms than were previously necessary.

Big Data has been characterized in several ways: as unstructured (Jurney, 2013), NoSQL (Moniruzzaman & Hossain, 2013; Xu et al., 2016), key-indexed, text, information-based (Tien, 2013), and so on. In view of this complexity, BDA requires a more comprehensive approach than traditional data mining; it calls for a unified methodology that can accommodate the velocity, veracity, and volume capacities needed to facilitate the discovery of information across all data types (Canada Health Infoway, 2013). Therefore, it is important to distinguish the various data types, because they affect the methodology of information retrieval (Alder-Milstein & Jha, 2013; Jurney, 2013; Tien, 2013).

There are also many recent studies of BDAs in healthcare defined according to technologies used, like Hadoop/MapReduce (Baro et al., 2015; Seo et al., 2016). Mostly, in the current literature, BDA can be defined partly in terms of technologies and their application to Big Data, especially for healthcare. Based within a framework or platform, it is the process used to extract knowledge from sets of Big Data (Hansen, Miron-Shatz, Lau, & Paton, 2014). BDA research has become an important and highly innovative topic across many and varied disciplines (Agrawal et al., 2012; Garrison, 2013; Shah & Tenbaum, 2012). The life sciences and biomedical informatics have been among the fields most active in conducting Big Data research (Liyanage et al., 2014). For example, the U.S. National Institutes of Health (NIH) has made data from its 1000 Genomes project (the world's largest collection of human genetic information) publicly available for analysis through the Amazon's Web Services (AWS) cloud-computing platform (Peek, Holmes, & Sun, 2014). There is also access to cloud resources via Canada National Research Council, who has investigated in millions to implement and maintaining cloud computing services for high performance computing research (NSERC, 2016). The NIH-sponsored Big Data to Knowledge (BD2K) initiative, with a budget of $656 million, funds research in biomedical informatics with a view to facilitating discovery (National Institutes of Health, 2014). Bateman and Wood (2009) used AWS to assemble a human genomics database with 140 million individual reads; the BDA for this database involved both alignment using a sequence search and alignment via a hashing algorithm (AWS, 2016). Unfortunately, very few studies of the application of BDA methodologies to the analysis of healthcare data have been published.

2.2 Big Data in Healthcare

In the 21st century, major advances in the storage of patient-related data have already been seen. There are many application types from research and development, public health and evidence-based medicine to genomic analysis and monitoring (Table 1). The advent of Big Data has affected many aspects of life, including biology and medicine because of available data (Martin-Sanchez & Verspoor, 2014). With the rise of the so-called "omics" fields (genomics, proteomics, metabolomics, and others), a tremendous amount of data related to molecular biology has been produced (Langkafel, 2016; Li, Ng, & Wang, 2014). Meanwhile, in the field of clinical healthcare, the transition from paper to EHR systems has also led to an exponential growth of data in hospitals (Freire et al., 2016). In the context of these developments, the adoption of BDA techniques by large hospital and healthcare systems carries great promise (e.g., Jee & Kim, 2013): not only can BDA improve patient care by enabling physicians, epidemiologists, and health policy experts to make data-driven decisions, but it can also save taxpayers' money by medical data mining, such as drug dispensing (Yi, 2011). Kayyali et al. (2013) estimated that the application of BDA to the U.S. healthcare system could save more than $300 billion annually. Clinical operations and research and development are the two largest areas for potential savings: $165 billion and $108 billion, respectively (Manyika et al., 2014).

Table 1. Big Data applications related to specific types of applications using big data.

Application Type	Description	Healthcare Application of Big Data
R&D	Predictive modeling and statistical tools (Hospital) (Grossglauer & Saner, 2014; Hansen, Miron-Shatz, Lau, & Paton, 2014; Nelson & Stragger, 2014; Manyika, Chui, Bughin, Brown, Dobbs, Roxburgh, & Hung, 2014)	-targeted R&D pipeline in drugs and devices -clinical trial design and patient recruitment to better match treatments to individual patients, thus reducing trial --failures and speeding new treatments to market -follow-on indications and discover adverse effects before products reach the market.
Public Health	Disease patterns and surveillance (larger than hospital level application) (Liyanange, de Lusignan, Liaw, Kuziemsky, Mold, Krause, Fleming, & Jones, 2014)	- targeted vaccines, e.g., choosing the annual influenza strains identify needs, provide services, and predict and prevent crises, especially for the benefit of populations
Evidence-based Medicine	Diagnosis and treatment (Hospital level)(Langkafel, 2016; Martin-Sanchez & Verspoor, 2014; Mohammed, Far, & Naugler, 2014; Rallapalli,	-combine and analyze a variety of structured and unstructured data-EMRs, financial and operational data, clinical data, and

	Gondkar, & Ketavarapu, 2016; Song & Ryu, 2015; Sun & Reddy, 2013; Tsumoto, Hirano, Abe, & Tsumoto, 2011)	genomic data to match treatments with outcomes, predict patients at risk for disease or readmission and provide more efficient care
Genomic Analysis	Gene sequencing and DNA match to treatment (individual to larger than hospital level application)(McKenna, Hanna, Banks, Sivachenko, Cibulskis, Kernytsky, Garimella, Altshuler, Gabriel, & Daly, 2010; Karim, Jeong, & Choi, 2012; Saunders, Miller, Soden, Dinwiddie, Noll, Alnadi, et al. 2012; Huang, Tata, & Prill, 2013; Wang, Goh, Wong, & Montana, 2013)	-make genomic analysis a part of the regular medical care decision process and the growing patient medical record
Device/remote Monitoring	Real-time data (Hospital Applied) (Miller et al., 2015; Twist et al., 2016; Nguyen, Wynden, & Sun, 2011)	-capture and analyze in real-time large volumes of fast-moving data from in-hospital and in-home devices, for safety monitoring and adverse prediction
Patient Profile Analytics	- apply advanced analytics to patient profiles (hospital) (Freire, Teodoro, Wei-Kleiner, Sundsvall, Karlsson, & Lambrix, 2016; Frey, Lenert, & Lopez-Campos, 2014; Wassan, 2014)	-identify individuals who would benefit from proactive care or lifestyle changes, for example, those patients at risk of developing a specific disease (e.g., diabetes) who would benefit from preventive care

The phenomenon of Big Data in healthcare of Health Big Data (cf. Kuo et al., 2014) began to attract scholarly interest in 2003, and since 2013 there have been more than 50 articles published annually (Baro et al., 2015). "Research has focused mainly on the size and complexity of healthcare-related datasets, which include personal medical records, radiology images, clinical trial data submissions, population data, human genomic sequences, and more. Information-intensive technologies, such as 3D imaging, genomic sequencing, and biometric sensor readings" (Baro et al., 2015; Foster, 2014), are helping to fuel the exponential growth of healthcare databases. And it is apparent that bioinformatics is leading the way with very few Big Data application of hospital systems. Nevertheless, databases are getting cheaper and increasing in size: data recorded in the U.S. healthcare system reached 150 Exabytes by 2011. On a global scale, digital healthcare data was estimated at 500 Petabytes in 2012 and is expected to reach 500 Exabytes by 2020 (Sun & Reddy, 2013). A single healthcare consortium, Kaiser Permanente

(which is based in California and has more than nine million members), is believed to have between 26.5 and 44 petabytes of potentially rich data from EHRs, including images and annotations (Freire et al., 2016).

Given its size, complexity, distribution, and exponential growth rate, Big Data in healthcare is very difficult to maintain and analyze using traditional database management systems and data analysis applications (Kuo et al., 2014). Health data includes "structured EHR data, coded data, . . . , semi-structured data . . . , unstructured clinical notes, medical images . . . , genetic data, and other types of data (e.g., public health and behavior data)" (Freire et al., 2016, p. 5). Moreover, "the raw data is generated by a variety of health information systems such as Electronic Health Records (EHR). . . , Computerized Physician Order Entry (CPOE). . . , Picture Archiving Communications System . . . , Clinical Decision Support Systems . . . , and Laboratory Information Systems used in . . . distributed healthcare settings such as hospitals, clinics, laboratories and physician offices" (Madsen, 2014, p. 41-54; Sakr & Elgammal, 2016).

Big Data in healthcare holds the promise of supporting a wide range of functions and users' contexts (Kuziemsky et al., 2014), including clinical decision support, disease surveillance, and population health management. BDA techniques can be applied to the vast amount of existing but currently unanalyzed patient-related health and medical data, providing a deeper understanding of outcomes, which then can be applied at the point of care (Ashish, Biswas, Das, Nag, & Pratap, 2013; Kubick, 2012; Sakr & Elgammal, 2016; Schilling & Bozic, 2014) and visualizations (Caban & Gotz, 2015). More specifically, BDA can potentially benefit healthcare systems in the following ways (among others): analyzing patient data and the cost of patient care to identify clinically effective and cost-effective treatments; advanced analytics (e.g., predictive modelling) to patient profiles in order to proactively identify who benefits the most from preventative care or lifestyle changes; broad-scale disease profiling and surveillance so as to predict onset and support prevention initiatives; facilitating and expediting billing authorization; accuracy and consistency of insurance claims for possible fraud; establishment of new revenue streams through the provision of data and services to third parties (for example, assisting pharmaceutical companies in identifying patients for inclusion in clinical trials) (Langkafel, 2016; Sakr & Elgammal, 2016; Raghupathi & Raghupathi, 2014).

Certain improvements in clinical care can be achieved only through the analysis of vast quantities of historical data, such as length of stay (LOS); choice of elective surgery; benefit or lack of benefit from surgery; frequencies of various complications of surgery; frequencies of other medical complications; degree of patient risk for sepsis, MRSA, *C. difficile*, or other hospital-acquired illness; disease progression; causal factors of disease progression; and frequencies of co-morbid conditions. In a study by Twist et al. (2015), the BDA-based genome-sequencing platform Constellation was successfully deployed at the Children's Mercy Hospital in Kansas City (Missouri, US) to match patients' clinical data to their genome sequences, thereby facilitating treatment (Saunders et al., 2012). In emergency cases, this allowed the differential diagnosis of a genetic disease in neonates to be made within 50 hours of birth. Improvement of the platform using Hadoop reduced the time required for sequencing and analysis of genomes from 50 to 26 hours (Miller et al., 2015). Unfortunately, this is one real-time implementation of only a few published studies of BDA platforms being used by healthcare providers to analyse hospital and patient data.

The use of Big Data in healthcare presents several challenges. The first challenge is to select appropriate statistical and computational method(s). The second is to extract meaningful information for meaningful use. The third is to find ways of facilitating information access and sharing. A fourth challenge is data reuse, insofar as "massive amounts of data are commonly collected without an immediate business case, but simply because it is affordable" (Madsen, 2014, p. 43). Finally, another challenge is false knowledge discovery: "exploratory results emerging from Big Data are no less likely to be false" (Nelson & Stragger, 2014, p. 32) than reporting from known data sets. In cancer registries, biomedical data are now being generated at a speed much faster than researchers can keep up with using traditional methods (Brusic & Cao, 2010). IBM Watson has demonstrated that large cancer registries, DNA markers, and patient prescriptions can be matched using this BDA tool to generate treatments specific to a person's DNA (Canada Health Infoway, 2013; Langkafel, 2016; Maier, 2013).

Matching medical treatments to patients' DNA has proven to be a valuable bioinformatics approach to therapy for certain diseases; however, deriving the appropriate relationships is challenging (Alder-Milstein & Jha, 2013). Without fundamental changes to documentation methods in health informatics, the ability of data mining to eliminate systematic errors is limited (Nelson & Staggers, 2014). However, an advantage of BDA platforms is that data can be easily aggregated. Aggregation of multiple terabytes of Big Data via Hadoop and MapReduce has proven effective in the area of social media (Jurney, 2013) at a magnitude 200 times larger than Kaiser Permanente's digital health records (Pearlstein, 2013). But finding patterns in the data can be difficult. There are also interoperability issues across different data sources that might invalidate pattern discovery; standards of interoperability (and data integration) in text, images, and other kinds of data are low in healthcare by comparison with other sectors (Langkafel, 2016; Swedlow, Goldberg, Brauner, & Sorger, 2003). An exception is medical imaging, which has been largely standardized (albeit using several competing lexicons), through the use of medical imaging (DICOM), HL7 transactions, and health information exchange formats (Langer & Bartholmai, 2011; Swedlow et al., 2003).

Some literature, especially in relation to the complex patterns of health informatics, suggests that data mining is one of many "mining" techniques. For example, there is text mining (e.g., Dai, Wu, Tsai, & Hsu, 2014; Debortoli, Muller, & vom Brocke, 2014), which some studies suggest is very different from data mining (Apixio Inc., 2013; Cios, 2001; Informatics Interchange, 2012; Rindflesch, Fiszman, & Libbus, 2005). Text mining is usually considered a subfield of data mining, but some text mining techniques have originated in other disciplines, such as information retrieval, information visualization (Lavarc et al., 2007), computational linguistics, and information science (Cios, 2001; Rindflesch et al., 2005; Wickramasinghe et al., 2009). There are a few studies in healthcare that use R with Hadoop for programming statistical software and visualizations (Baro et al., 2015; Das, Sismanis, & Beyer, 2010; Huang, Tata, & Prill, 2013; RHIPE, 2016); however, there have been major programming advances via GitHub (RHadoop and MapR, 2014). In practice, graph processing tasks can be implemented as a sequence of chained MapReduce jobs that involve passing the entire state of the graph from one step to the next (Sakr & Elgammal, 2016). However, this mechanism is not suited for graph analysis and leads to inefficient performance because of communication overhead, associated serialization overhead, and the additional requirement of coordinating the steps of a chained

MapReduce over Hadoop's HDFS (Sakr, Liu, & Fayoumi, 2013). Other disciplines displace mining aspects further into images, privacy, lean or operational patient flow, and overall information-based analytics (Chen et al., 2005). Moreover, there are differences among descriptive, analytical, prescriptive, and predictive analytics in healthcare (Podolack, 2013). Even family history has its own unique mining process in EHRs (Hoyt et al., 2013). Data mining can also be qualitative, in which case interactions with medical professionals play an indispensable role in the knowledge discovery process (Castellani & Castellani, 2003) and even between bioinformatics and health informatics (Miller, 2000).

2.3 Big Data Technologies and Platform Services

Big Data technologies fall into four main categories: high-performance computing, data processing, storage, and resource/workflow allocator, like Hadoop/MapReduce framework (Dunning & Friedman, 2010; Holmes, 2014; Khan et al., 2014; Mohammed et al., 2014; Yao, Tian, Li, Tian, Qian, & Li, 2015). A high-performance computing (HPC) system is usually the backbone or framework of a BDA platform, for example, IBM's Watson and Microsoft Big Data solutions (Jorgensen et al., 2014). An HPC system consists of a distributed system, grid computing, and a graphical processing unit (GPU). Moreover, in a distributed system, several computers (computing nodes) can participate in data processing a large volume and variety of structured, semi-structured, and/or unstructured data. A grid computing system is a distributed system employing resources to allocate in multiple locations (e.g., CPUs, storage of computer systems across network, etc.), which enables processes and configurations to be applied to any task in a flexible, continuous, and inexpensive manner (Chen & Fu, 2015). GPU computing is well-adapted to the throughput-oriented workload problems that are characteristic of large-scale data processing. Parallel data processing can be handled by GPU clusters (Dobre & Xhafa, 2014; Karim & Jeong, 2011). However, "GPUs have difficulty communicating over a network, cannot handle virtualization of resources, and using a cluster of GPUs to implement the most commonly used programming model (namely, MapReduce) can present some challenges…" (Mohammed et al., 2014).

In the quest for the most effective BDA platform, distributed systems appear to be the wave of the future. It is important to understand the nature of a distributed system as compared to conventional grid computing, which can also be applied to supercomputing and high-performance computing, which does not mean data mining or big data tools (Lith & Mattson; Maier, 2013). A distributed computing system can manage hundreds or thousands of computer systems, each of which is limited in its processing resources (e.g., memory, CPU, storage, etc.). By contrast, a grid computing system makes efficient use of heterogeneous systems with optimal workload management servers, networks, storage, and so forth. Therefore, a grid computing system supports computation across a variety of administrative domains, unlike a traditional distributed computing system.

The kind of computing with which most people are familiar uses a single processor (desktop personal computers or laptops for example), with its main memory, cache, and local disk (that can be referred to as a computing node). In the past until now, applications used for parallel processing for large scientific or statistical calculations employed special-purpose parallel computers with many processors and specialized hardware. However, the prevalence of large-scale web services and IBM-Apache WebSphere has encouraged a turn toward distributed

computing over enterprise systems or platforms: that is, installations that employ thousands of computing nodes operating more or less independently (e.g., Taylor, 2010; Langkafel, 2016). These computing nodes are off-the-shelf hardware, which greatly reduces the cost of distributed systems (compared to special-purpose parallel machines). To meet the needs of distributed computing, a new generation of programming frameworks attempt to take advantage of the power of parallelism while avoiding its pitfalls, such as the reliability problems in the use of hardware of thousands of independent components that any can fail at any time. For example, the Hadoop cluster, with its distributed computing nodes and connecting Ethernets, runs jobs controlled by a master node (known as the NameNode), which is responsible for chunking data, cloning it, sending it to the distributed computing nodes (DataNodes), monitoring the cluster status, and collecting or aggregating the results (Taylor, 2010). It, therefore, becomes apparent that not only the type of architecture and computing system is important for the establishment of a BDA platform but also the inherent and customizable processes in the data brokerage.

A number of high-level programming frameworks have been developed for use with distributed file systems; MapReduce, a programming framework for data-intensive applications proposed by Google, is the most popular. Borrowing ideas from functional programming, MapReduce enables programmers to define Map and Reduce tasks to process large sets of distributed data; thus, allowing many of the most common calculations on large-scale data to be performed on computing clusters efficiently and in a way that is tolerant of hardware failures during compaction/computation. Distributed computing using MapReduce and Hadoop represent a significant advance in the processing and utilization of Big Data in the healthcare field (Mohammed et al., 2014; Sakr & Elgammal, 2016). However, MapReduce is not suitable for online transactions or streaming (Sitto & Presser, 2015), and, therefore, the system architecture compatible with the MapReduce programming might require a great deal of customization.

The key strengths of the MapReduce programming framework are its high degree of parallelism of inherent tasks combined with its simplicity and its applicability to a wide range of domains (Lith & Mattson, 2010; Mohammed et al., 2014). The degree of parallelism depends on the size of the input data (Lith & Mattson, 2010). The Map function processes the inputs (e.g., key1, value1), returning other intermediary pairs (e.g., key2, value2). Then the intermediary pairs are grouped together according to their keys (Jorgensen et al., 2014). The Reduce function outputs new KV pairs (e.g., key3, value3). High performance is achieved by dividing the processing into small tasks that can be run in parallel across tens, hundreds, or thousands of nodes in a cluster (Lith & Mattson, 2010). Programs written in a functional style (in Java code) are automatically parallelized and executed by MapReduce, making the resources of large distributed systems available to programmers without any previous experience with parallel or distributed systems. Distributed systems that employ MapReduce and Hadoop have two advantages: (1) reliable data processing via fault-tolerant storage methods that replicate computing tasks and clone data chunks on different computing nodes across the computing cluster; and, (2) high-throughput data processing via a batch processing framework and the HDFS (Taylor, 2010). Thus, it is apparent that not only is Hadoop and MapReduce compatible but also the inherent processes of MapReduce are important for the platform to perform well as the volumes increase.

Designed by Apache, and creator Doug Cutting (who also originated Lucene), Hadoop is an open-source software implementation of the MapReduce framework for running applications on

large clusters of computers. Hadoop provides both distributed storage and computational capabilities. "Hadoop was first developed to fix a scalability issue affecting *Nutch*, an open-source crawler and search engine that uses the MapReduce and big-table methods developed by Google" (Madsen, 2014, page 44). Hadoop employs a distributed master-slave architecture that consists of the HDFS for storage and the MapReduce programming framework capabilities. A large and growing range of Big Data technologies related to Hadoop are emerging (Table 2). Open-source software packages for use with the HDFS are constantly being updated with newer versions and increasing in number between 30-40 packages that can be installed on computing nodes. Sometimes newer versions are a beta-test for quick turnaround of newer more stabilized versions and cannot be used with certain Apache Hadoop versions or they can improve their compatibility with the Hadoop backbone on the node. The HDFS stores data on computing nodes, providing a very high-aggregate bandwidth across the cluster. Its storage and computational capabilities scale with the addition of computing nodes, and can reach volume sizes in the petabytes on clusters with thousands of nodes, Yahoo currently still has the largest at ~3800 (Gray, 2008).

Table 2. Big Data Technologies using Hadoop with possible applications in healthcare.

Big Data Technologies	**Description**	**Purpose**	**Applied in Healthcare**
Hadoop Distributed File System (HDFS)	The Hadoop Distributed File System (HDFS) is the place in a Hadoop cluster where you store data (Apache Hadoop, 2016). Built for data-intensive applications, the HDFS is designed to run on clusters. HDFS is optimized for high performance, read-intensive operations, and is resilient to failures in the cluster. It does not prevent failures, but likely to lose data, because HDFS by default makes multiple copies of each of its data blocks (White, 2015; Hadoop, 2016).	High capacity, fault tolerant, inexpensive storage of very large datasets (Jurney, 2013)	Yes (Langkafel, 2014; Karim et al., 2013; Taylor, 2010; Sakr and Elgammal, 2016)
MapReduce	MapReduce was the first and is the primary programming framework for developing applications in Hadoop. You'll need to work in Java to use MapReduce in its original and pure form (Dean & Ghemawt, 2010).	A programming paradigm for processing big data.	Yes (Taylor, 2010; Raghupathi & Raghupathi, 2014)

Hadoop	Fully integrated, and linkage between two technologies: HDFS and MapReduce (Mohammed, Far, & Naugler, 2014).	Processing	Yes (Taylor, 2010)
Spark	MapReduce is the primary workhorse at the core of most Hadoop cluster. While highly effective for very large batch-analytic jobs, MapReduce has proven to be suboptimal for applications like graph analysis that require iterative processing and data sharing. Three core areas: resilient distributed dataset (RDD), transformation, and action (Karau, Konwinski, Wendell, & Zatharia, 2015).	Processing\storage	No
Cassandra	Key-value datastores are a common fixture in any big data system because they are easy to scale, quick, and straightforward to work with. Cassandra is a distributed key-value database designed with simplicity and scalability in mind (Lith & Mattson, 2010).	Key-value store	No (Sakr & Elgammal, 2016)
HBase	Many attributes of the data, but each observation only has a few of them. HBase is a NoSQL database system included in the standard Hadoop distributions. It is a key-store, logically. This means that rows are defined by a key, and have associated with them a number of bins (or columns) where the associated values are stored (HBase, 2016). The only data type is the byte string. Physically, groups of similar columns are stored together	NoSQL database with random access	Yes (Mohammed, Far & Naugler, 2014; Chang et al., 2013; Moniruzzaman & Hossain, 2013)

	in columns families. Most often, HBase is accessed via Java code, but APIs exist for using HBase with Pig, Thrift, Jython (Python based), and others. HBase is not normally accessed in a MapReduce fashion. It does have a shell interface for interactive use.		
Apache Solr	While Solr is able to use the Hadoop Distributed File System to store data, it is not truly compatible with Hadoop and does not use MapReduce or Yarn to build indexes or respond to queries. There is a similar effort named Blur to build a tool on top of the Lucene framework that leverages the entire Hadoop stack (Seo, Kim, & Choi, 2016).	Document Warehouse	No
Lucene and Blur	Blur is a tool for indexing and searching text with Hadoop. Because it has Lucene (a very popular text-indexing framework) at its core, it has many useful features, including fuzzy matching, wildcard searches, and paged results. It allows you to search through unstructured data in a way that would otherwise be very difficult.	Document Warehouse	No, not in healthcare but upcoming development and patents (Seo, Kim, & Choi, 2016).
MongoDB	MongoDB is a document-oriented database, the document being a JSON object. In relational databases, you have tables and rows. In MongoDB, the equivalent of a row is JSON document, and the analog to a table is a collection, a set of JSON documents (Maier,	JSON document-oriented database	Yes (Mohammed, Far, & Naugler, 2014)

	2013).		
Hive	The goal of Hive is to allow SQL access to data in the HDFS (Hive, 2016). The Apache Hive data-warehouse software facilities querying and managing large datasets residing in HDFS. Hive defines a simple SQL-like query language, called HQL that enables users familiar with SQL to query the data (Hive HBase, 2016).	Data Interaction	No (Xu, Shi, Chen, Zhang, Fu, & Liu, 2016)
Spark SQL	Spark outperforms Hive.	SQL access to Hadoop data	No
JSON	JSON is becoming common in Hadoop because it implements a key-value view of the world.	Data description and transfer	Yes (Sakr & Elgammal, 2016)
ZooKeeper	Hadoop and HDFS are effective tools for distributing work across many machines. ZooKeeper is not intended to fill the space of HBase or any other big data key-value store. In fact, there are protections built into ZooKeeper to ensure that folks do not attempt to use it as large data store. ZooKeeper is, however, just right when all you want to do is share a little bit of information across your environment (ZooKeeper, 2016).	Coordination	Yes (Taylor, 2010)
YARN (Yet Another Resource Negotiator)	Hadoop resource allocator. It is a resource-management platform responsible for managing compute resources in clusters and using them for scheduling of users' applications (Dunning & Friedman, 2010).	Resource allocator	Yes (Taylor, 2010)
Oozie	Hadoop's workflow scheduler (White, 2015)	A workflow scheduler to	No

		manage complex multipart Hadoop jobs	
Pig	Pig is translated or compiled into MapReduce code and it is reasonably well optimized so that a series of Pig statements do not generate mappers and reducers for each statement and then run them sequentially (Norberg, Bhatia, Wang, & Wang, 2013).	High-level data flow language for processing data.	No
Storm	Many technologies in the big data ecosystem, including Hadoop MapReduce, are built with very large tasks in mind. These systems are designed to perform work in batches, bundling groups of smaller tasks into larger tasks and distributing those large tasks (Maier, 2013).	Streaming ingest	Yes (Miller et al., 2014; Twist et al., 2016)

Hadoop differs from other distributed system schemes in its approach to handling data (e.g., Mohammed et al., 2014; Raghupathi & Raghupathi, 2014). A traditional distributed system requires repeated transmissions of data between clients and servers. This works fine for computationally intensive work, but for data-intensive processing, the data becomes too large to be easily transmitted. Hadoop solves this problem by moving code to data instead of data to code. The client (NameNode) sends only MapReduce programs to be executed, and these programs are usually small (often only kilobytes in size). Importantly, the move-code-to-data approach is applied within the Hadoop cluster itself. Data is broken up and distributed across the cluster, and computation on a chunk of data takes place, as far as possible, on the same machine in which that chunk of data resides.

Chang, Liao, Fan, and Wu (2014) state that the HDFS is designed to deal with data for building a distributed data center. The HDFS uses data duplicates to enhance reliability. However, data duplicates need extra storage space and hence a larger investment in infrastructure. Deduplication techniques can improve efficiency in the use of storage space (Change et al. 2014). Even efficient configuration at one point in time, the number of software versions and complexity of software in Hadoop's ecosystem increasing, there are more software and big data platforms applied to bioinformatics and genetics than hospital systems, which currently use very few users and even less applied to real-time utilization. Relatively new software like Spark (est. 2013) and Drill (est. 2015) has not been fully tested for clinical databases and systems (e.g., Sitto & Presser, 2015). There is a "showdown" in the IT software world between these two products and that depends on the type of SQL on Hadoop and goals for end user computing and usability

(Scott, 2015). Drill uses ANSI-SQL:2003 and it is similar SQL to MS SQL Server, MySQL, or Oracle, and, therefore, easy to utilize because it conformance and adherence to standardized SQL (Dunning, Friedman, Shiran, & Nadeau, 2016). On the other hand, Spark requires the programmer/analyst to write the code in Python, Scala or Java to execute SQL that can be highly customized (Karau, Konwinski, Wendell, & Zatharia, 2015). Additionally, Apache Drill technologies can capture data schemas in real time that allows for high performance and optimization on the fly for greater efficiency. Even more so, that due to Drill's inherent technologies to optimize on the fly and leveraging file system permissions more than Spark, it can provide enhanced privacy/security of the data and can be compliant with HIPAA use cases (Scott, 2015). However, very little is known about the usability differences of aligned technologies with Hadoop that could contribute to their utilization in this context of big data platforms or frameworks.

In considering the design and implementation of BDA systems for use in the field of healthcare, the basic premise is to construct a platform capable of compiling diverse clinical data from diverse sources and querying large volumes of data quickly. A simulated Big Data environment is required to test these capabilities. In such a simulation, however, no algorithms for real data can be verified because they have not been used in Hadoop/MapReduce before testing. While a number of studies in genetics have compared libraries to libraries (e.g., Miller et al., 2016; Twist et al., 2015), unfortunately no research into the formation and querying of databases for entire hospital or healthcare systems has been conducted to date. Large volumes of healthcare data are generated and queried every day, but these tasks have yet to be facilitated by Big Data technologies. The present investigation is, therefore, the first to test the application of BDA on the (simulated) database of a large healthcare system. Hence, a simulation to generate the data would be required to then run queries over the platform to ensure that the data and simplified algorithms are deploying and running Hadoop and MapReduce in their programmed framework.

There are many alternative solutions for databases in Big Data platforms; choice of the best solution depends on the nature of the data and its intended use (e.g., Lith & Mattson, 2010). In practice, while many systems fall under the umbrella of NoSQL systems and are highly scalable (e.g., Tauro, Aravindth, & Shreeharsha, 2012), these storage types are quite varied. However, each comes with its unique sets of features and value propositions (Sakr, Liu, Batista, & Alomari, 2011). For example, the key/value (KV) data stores represent the simplest model of NoSQL systems: they pair keys to values in a very similar fashion to how a map (or hashtable) works in any standard programming language. Various open-source projects have been implemented to provide key-valued NoSQL database systems; such projects include Memcached, Voldemort, Redis, and Basho Riak (Sakr & Elgammal, 2016). Another category of NoSQL systems is document-oriented database stores. In these systems, a document is like a hash, with a unique ID field and values that may be any of a variety of types, including more hashes. In particular, documents can contain nested structures, so they provide a high degree of flexibility, allowing for variable domains such as MongoDB and CouchDB (Sakr & Elgammal, 2016). Finally, NoSQL Graph databases are yet another category of systems; they excel in handling highly interconnected data. These other categories could be used for hospital data; however, in this study HBase was chosen as the database type and technology because it simplified the emulation of the columns using the metadata in each column rather than the data types and the actual relationships among the data. HBase also has a dynamic schema that can be

uploaded via other Apache applications; therefore, the schema can be changed and tested on the fly. If HBase had not been used, more complex data models would have been needed to map over the Hadoop/MapReduce framework. Another benefit of using HBase is that further configurations can be accomplished for multi-row transactions using a comma-separated value (*.csv*) flat file (Zhang, 2010). In fact, always the row and column sizes need to be minimized. The KV class is the heart of data storage in HBase, which provides inherent encryption. When HBase was designed and implemented, RowKey, ColumnFamily, and RowColumns in HBase needed to have names as short as possible, since all of them were embedded within the KV instance (HBase, 2016). The longer these identifiers are, the bigger the KV of data storage in HBase will become; therefore, identifier length was standardized in this study as the minimum required depicting the data profile. Problems appeared while creating corresponding RowKeys in HBase. The ingestions were not evenly distributed, and the increasing keys in a single region may have contributed to the Reducer being slow (Sakr & Elgammal, 2016).

3. Materials and Methods

3.1 High Performance Computing Infrastructure

This study used a high performance computing architecture by WestGrid to install the open-source software of Hadoop packages, build the configurations and run the tests. WestGrid computing facilities are part of Compute Canada's national platform of advanced-research capacity resources and collaboration distributed among several resource provider sites, with some degree of specialization at each site. High-performance networks connect WestGrid clusters and sites. WestGrid installation at University of Victoria (UVic) started in July 2010; it makes its computing and storage elements available to clients through various services.

The WestGrid computing facilities at UVic (University of Victoria, 2016) use the Hermes and Nestor system the Hermes and Nestor system consists of two clusters that share infrastructure, such as resource management, job scheduling, networked storage, and service and interactive nodes. Hermes and Nestor share login nodes: hermes.westgrid.ca and nestor.westgrid.ca (hermes.westgrid.ca and nestor.westgrid.ca are aliases for a common pool of head nodes named litaiNN.westgrid.ca). Nestor is a 2304-core capability cluster geared towards parallel jobs. It consists of 288 IBM iDataplex servers with eight 2.67 GHz Xeon x5550 cores and 24 GB of RAM. Data is shared between nodes and the GPFS file system using a high-speed InfiniBand interconnect. Hermes is a 2112-core capacity cluster geared towards serial jobs. It consists of 84 nodes having eight cores each and 120 nodes with 12 cores each.

Hermes is deployed in two phases: the first phase consists of 84 IBM iDataplex servers with eight 2.67 GHz Xeon x5550 cores and 24 GB of RAM. These nodes use bonded gigabit Ethernet to get data from NFS and GPFS file systems. The second phase consists of 120 Dell C6100 compute nodes, each with 12 x 5650 at 2.67 GHz and 24 GB of RAM. These nodes use a 10:1 blocking InfiniBand fabric (low latency and high bandwidth) for their cluster network.

WestGrid provides several types of computing systems, since different users' programs have different requirements. Users can access whichever system best fits their needs, regardless of physical location. The main WestGrid computational clusters are accessed by submitting batch

job scripts from a login server. Wall time and accessing nodes while running a job are very restricted. Only the administrator can access the node to kill a job but even then the data cannot be accessed.

Access to WestGrid resources requires a WestGrid account, which requires registration and sponsorship with Compute Canada. Once you have a WestGrid ID, the cluster's interactive nodes can be accessed by *ssh* to hermes.westgrid.ca or nestor.westgrid.ca (via MobaXterm or Putty for example). MobaXterm allows for log files to be shown in a Windows-like directory and file dragged and dropped on the desktop. Currently, there are two interactive nodes and round-robin Domain Name Server (DNS) is used for the initial load-balancing access. At a server hardware and secure login level, lightweight directory access protocol (LDAP) on the Deployment Managers (DMs) offers secure access to dedicated local disk storage. Access to worker nodes is restricted to the user running the job on them. That is, no other user can alter the job or even access the node if the current user has a job running. Additionally, there is a layered-security approach in which access to WestGrid follows a strict and secure policy of approving accounts. Two approvals are usually required: one by the sponsor of the account and the other by the administrators. The sponsors (such as faculty members) have to be eligible for Canadian Foundation for Innovation (CFI) funding in order to have an account. From a technical perspective, the administrators follow approved internal security policies to keep the clusters secure.

The systems are designed for high-performance and advanced-research computing and include clusters with fast interconnection and shared memory systems. Serial programs run on one CPU or core on a computing cluster node. Some researchers have serial programs that need to be run many times, using different parameter values. In such cases, multiple copies can be run simultaneously on a cluster by submitting as many jobs as are required. Parallel programs, on the other hand, may have multiple processes or threads running at the same time, so that installations need to communicate to carry out their tasks. In order of increasing demands, programs can run either on a cluster with a fast interconnection or on a shared-memory cluster, depending on how the programs are written (MPI, OpenMP, threads, etc.). How well a parallel program scales determines how many nodes of a cluster that program should be run on. Other factors will also affect the decision as to which cluster system to run on. Some of the determining factors include the amount of memory available (particularly the amount of memory per processor), the software that is installed, and restrictions due to software licensing. This study used both types to send job to ingest file and also to run the Hadoop/MapReduce framework.

TORQUE was evolved from software called Portable Batch System (PBS) and utilized by WestGrid to allow users to run their own jobs. PBS script is a script composed of two sections: one section specifying the requirement of the job (resources needed to process) and one run on the worker nodes. The requirements can be specified in the command *qsub*. To submit the script *diffuse.pbs* to the batch job handling system, the *qsub* command, *qsub –I*, is used. When *qsub* processes the job, it assigns it a job ID and places it in a queue to await execution. If a job is expected to take longer than the default time limit (typically three hours) or use more than the default memory, additional arguments may be added to the qsub command line. If diffuse is a parallel program, you also have to specify the number of nodes on which it is to run. For example: *qsub –l walltime=72:00:00,mem=1500mb,nodes=4* (i.e., 72 hours is the maximum).

Example (for this study to run Hadoop-MapReduce):

```
#!/bin/bash –l
#PBS -N Hadoop_xple
#PBS -q nestor-test
#PBS -l walltime=01:00:00
#PBS -l nodes=2:ppn=8
#PBS -l mem=16gb
#PBS -j oe

cd $PBS_O_WORKDIR
module load Hadoop/2.6.2
which java
which hdfs
setup_start-Hadoop.sh
hdfs dfsadmin -report
hdfs dfs -mkdir /data
hdfs dfs -put allpg2701.txt/data
hdfs dfs -ls /data
Hadoop jar $HADOOP_HOME/share/Hadoop/MapReduce/Hadoop-*-examples-*.jar wordcount
/data/pg2701.txt/wordcount-output
hdfs dfs -cat /wordcount-output/part-r-00000
hdfs dfs -ls /wordcount-output
hdfs dfs -get /wordcount-output ./
clean_stop-Hadoop.sh
```

In the above example, "word count" is a test example. It is a well-known test and recommended by Apache to test the configurations of MapReduce to split the text in words in its Java calling the *StringTokenizer* method and other methods provided in the tutorial (Apache Hadoop, 2016). Apache Hadoop open source members provide a large text file via the Moby Dick book that then Hadoop can count the words via the correctly customized Java code in MapReduce from tutorial. The performance is within a few minutes the "wordcount-output" will show the exact number of words. If it takes longer than a few minutes to generate the output, then the install of the Hadoop version and package configurations needs to be reviewed for errors and some parts of its default xml needs to be enabled or disabled.

Job scheduling in a batch system can be very complex, with many variables. Scheduling – *fairshare* – people with less usage have higher priority – is implemented. Scheduling software Moab allows a great deal of flexibility; while a small cluster serving one research group may use a very simple scheduler (possibly even a simple first-come that is first to log in to the node is the first-served policy at WestGrid), scheduling for a system the size of Hermes/Nestor serving various research groups at many different institutions is more complicated. Wall-clock time, sometimes called simply *walltime*, is the time used or expected to be used by a job, regardless of actual CPU cycles consumed. Most clusters will terminate a job whose actual wall time exceeds the expected wall time declared when the job was submitted. In this study, local disks were

purchased via VIHA's seed grant competition; therefore, most times the Hermes nodes were dedicated as Hadoop ingestion to HBase was continually run and tested.

3.2 Healthcare Big Data Analytics (HBDA) Framework

Hadoop/MapReduce framework was proposed to implement HBDA and analyze emulated patient data over a distributed computing system that is not currently used in acute patient care settings at VIHA and other health authorities in British Columbia and in Canada. The teamed collaboration between UVic, Compute Canada/WestGrid, and VIHA established the framework of the HBDA platform. It comprised innovative technologies like the Hadoop HDFS with MapReduce programming, and a NoSQL database, called HBase. The database construct is complex and had many iterations of development over the past three to four years. HBase is an open-source, distributed key-value (KV) store based on Google's BigTable (Chang, Dean, Ghemawat, Hsieh, Wallach, Burrows, Chandra, & Fikes, 2006) ─ persistent and strictly consistent NoSQL system using Hadoop Distributed File System (HDFS) for data storage. Furthermore, with all these technical components to construct the platform, the build also took into account the workflow at VIHA with their respective clinical reporting workgroups to form a database with the same metadata from real hospital datasets (Figures 5 and 6).

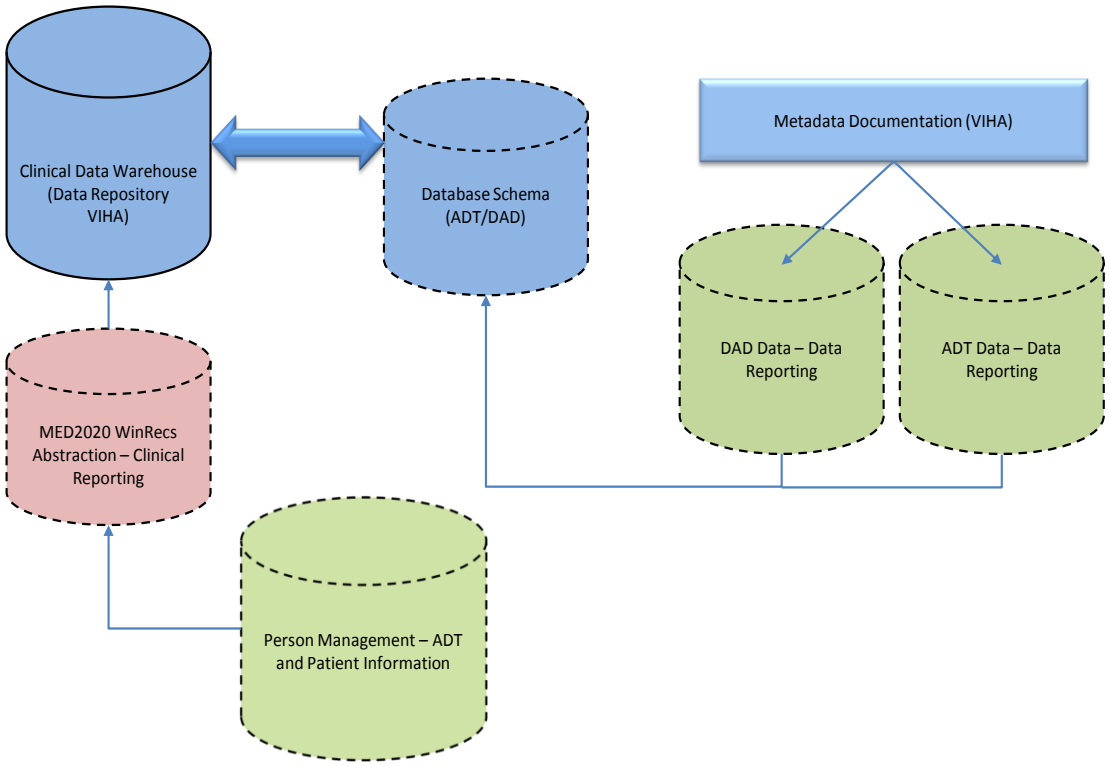

Figure 5. The workflow of abstracting the admission, discharge, and transfer (ADT) and Discharge Abstract Database (DAD) metadata profiles including workflow steps carried out on a regular basis by VIHA staff only. Med2020 WinRecs abstraction software is used to abstract data based on dictionaries and data standards, accordingly.

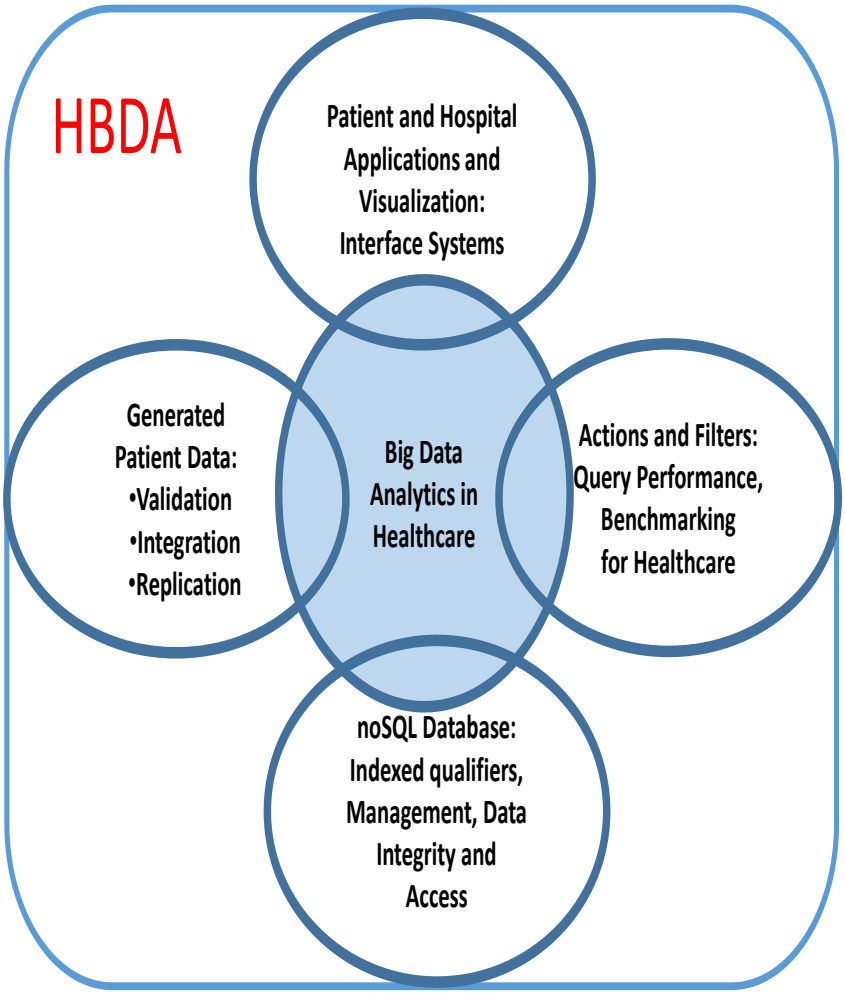

Figure 6. The main components of our Healthcare Big Data Analytics (HBDA) platform that were envisioned by stakeholders and derived from research team.

There are numerous reasons why the BDA platform did not use real patient data. Firstly, the process of Ethics and Research Capacity at VIHA for approval for the entire patient data of the hospital system is not possible without existing system and total security/privacy testing. Secondly, it is not possible to piece together summarized data specific to health outcomes because this data has already been summarized and no benefit to VIHA, and, therefore, Ethics will not approve. Thirdly, in a real setting the data will have to be moved or migrated off of the production architecture to avoid using and consuming network resources in use in the hospital. Fourthly, real data in the data warehouse at VIHA will require several months to review and develop the solution to use big data technologies, which is not available. Fifthly, the platform

will need to use the big data technologies to not only move the data but also make sure the data integrity and quality is maintained, which means that a non-production test in simulation can prove this is possible. Lastly, performance benchmarking of the platform needs to be gauged with the current data query tools at VIHA, which means that simulation at extremely large volume can prove high performance and evidence that this would be beneficial. Therefore, the study focus on a simulation was conducted with VIHA's Ethics and Research approval to use real metadata and exchange knowledge on how the ADT and DAD are constructed in current production.

There were many configurations and package components to include in the build, such as Apache Phoenix, Apache Spark, and Apache Drill, as well as Zeppelin and Jupyter Notebook interfaces. Furthermore, testing the platform required a proof-of-concept to implement the BDA platform in simulation before applying it to real patient data. The study aim was to effectively query 1-10 billion records of patient data, i.e., similar to the size and number of patient encounters stored in the VIHA over the past 50 years. With the help from health professionals and providers their current fast and reliable queries were revealed, as well as unknown and desired health trends, patterns, and associations of medical services with health outcomes were unveiled. The records comprise patient demographics; emergency care; admission/discharge/transfer; clinical activities; diagnoses; and, outcomes information.

Specifically, the methods and platform establishment was related to the study objectives:

- Establish a pipelined BDA process and configuration that meets patient data security, confidentiality, and privacy requirements and, ultimately, form a standard for similar studies in Canada;
- Execute BDA with a distributing filing/search system in a way that has not been performed previously on real patient data as a proof-of-concept of this process; and
- Propose a proof-of-concept solution for specific analytic challenges (in Section 1.1) and investigate previously unknown challenges and corresponding resolutions in the data process pipeline.
- The working hypothesis is that the proof-of-concept platform will exhibit slower processing for queries of data sets of three billion versus one billion versus 50 million patient encounters or rows, and slower performance for Apache Phoenix versus Apache Spark and Apache Drill, and slower for complex queries related to services and outcomes than for standard diagnosis, date-of-birth (DOB), age, and LOS.

The functional platform was tested for performance of data migrations or ingestions of HFiles via Hadoop (HDFS), bulkloads of HBase, and ingestions of HFiles to Apache Spark and Apache Drill. In this study performances were proof-of-concept testing using simulated data with the same replicated metadata and very large volume, but this did not consist of comparing performance of SQL (relational database) with NoSQL or different data models with real patient data. Furthermore, this study involved six the Hermes nodes (each node has 12 cores), which only 72 cores in total of the possible 4416 cores at WestGrid-UVic. The bulk of the methodology is generating the emulated data and queries with Hadoop configurations and other software, i.e., HBase, Spark and Drill. Most of the configurations were somewhat out-of-the-box defaults for the distributed filing and MapReduce in Java, Python and Scala to perform as expected;

therefore, the platform was established by Hadoop-MapReduce configurations to run and integrate with all other big data tools.

3.3 Data Source and Analytic Tools

To perform this study, and as a proof-of-concept investigation, the platform used six dedicated nodes (12 cores, 24GB of RAM, and eventually three 2TB-disks each were placed in the RAC as dedicated table space to 36TB plus 4-6TB additional space for a total of ~40TB) from the WestGrid supercomputing facility at UVic. The framework of the platform relied on the UVic WestGrid computer cluster to analyze 1-10 billion healthcare records stored in the VIHA clinical data warehouse. It also relied on VIHA's staff members from BI data warehouse, application platform, health informatics architecture, and clinical reporting teams to establish the requirements of the emulation of the DAD and ADT data, as well as to compare their workflows to clinical reporting and finding trends.

Initially, the platform was tested with real data from a publicly available annual (2005-2006) inventory of pharmaceuticals of a health authority in the Province that had 65,000 rows and 37 columns (~5MB). Results of the summarizing the inventory was tested over the platform and compared with known summarized data. Additionally, I2B2 data sets from the Northeastern US were acquired via their application and research ethics evaluation (https://www.i2b2.org/) to test the platform to query unknown results. However, all their data sets were small (<1GB) and were either summarized data for population health data or free text (e.g., Uzuner, South, Shen, & Duvall, 2011) from an example smaller snippet of progress reports in EHRs. Therefore, to validate the queries only the pharmaceutical data was used followed by ADT/DAD emulation at extreme volumes that represented the VIHA's data warehouse.

Access to the head node to install software and configuration aided in the implementation of a NoSQL Big Data database (in this case, HBase) and established the platform that will enable the BDA to simulate querying billions of healthcare records stored in VIHA's clinical data warehouse for future work. The proposed architecture relies on the way the WestGrid clusters are designed: a head node (a master deployment manager - DM) is used to access the cluster as well as to interact with the worker nodes via the Torque/ PBS resource manager (RM), which has agents (PBS moms) running on the worker nodes. The access to the DM is controlled via LDAP while access to worker nodes is restricted by batch jobs to only the user running the job and the administrator, who can kill the job but not access the data. This architecture allowed for deployment of a BDA platform on existing secure framework. It also permitted an agile BDA platform, in the sense that it can be launched and terminated under a PBS job.

3.4 Data Replication, Generation and Analytics Process

i. Interviews

Metadata is information about the data that is established in a system as a structured standard to record and retrieve information accurately. It is the structure of metadata that allows for data profiles (i.e., characteristics, sources, character lengths, etc.) to be established in a database, and in healthcare this means data is standardized effectively for patient records to be accurate when retrieved or viewed in an EHR. In the case of Island Health, the metadata of the Admission, Discharge, Transfer (ADT) system allows for patient data to be recorded when a patient is admitted to the hospital, assigned to a bed, and provided other medical services. The structure

itself is not proprietary to the system and does not contain any real patient data. However, due to the uniqueness of the system, consultation with Island Health is needed to mimic the metadata for patient outcomes so that queries on large databases can be constructed to perform valid analysis of the patient care and healthcare system.

To accomplish these objectives, Island Health's partial core metadata from ADT/ Discharge Abstract Database (DAD) systems was obtained via knowledge transfer in interviews with specific teams working at Royal Jubilee Hospital (RJH). Knowledge transfer was primarily completed by meeting the VIHA personnel in person (recorded and documented) and secondarily with brief discussions over the phone or asynchronously via email, and was conducted under VIHA-UVic Research Ethics' approval after interview scripts were constructed and mad available (Table 3). All the technological and programming requirements were done with WestGrid and their IT resources; this work has been documented via VIHA-UVic collaboration and new hardware established after winning a $5000 seed-grant annual research competition.

Table 3. Interview Scripts – Case Scenarios

Case 1: Uncontrolled Type 2 diabetes & Complex comorbidities: A.B. is a retired 69-year-old man with a 5-year history of type 2 diabetes. Although he was diagnosed in 1997, he had symptoms indicating hyperglycemia for 2 years before diagnosis. He had fasting blood glucose records indicating values of 118–127 mg/dl, which were described to him as indicative of "borderline diabetes." A.B. presented with uncontrolled type 2 diabetes and a complex set of comorbidities, all of which needed treatment. **Physical Exam:** Weight: 178 lb; height: 5′2″; body mass index (BMI): 32.6 kg/m2 Fasting capillary glucose: 166 mg/dl Blood pressure: lying, right arm 154/96 mmHg; sitting, right arm 140/90 mmHg Pulse: 88 bpm; respirations 20 per minute Eyes: corrective lenses, pupils equal and reactive to light and accommodation, Fundi-clear, no arteriolovenous nicking, no retinopathy Thyroid: nonpalpable Lungs: clear to auscultation Heart: Rate and rhythm regular, no murmurs or gallops Vascular assessment: no carotid bruits; femoral, popliteal, and dorsalis pedis pulses 2+ bilaterally Neurological assessment: diminished vibratory sense to the forefoot, absent ankle reflexes, monofilament (5.07 Semmes-Weinstein) felt only above the ankle **Lab Results:** Glucose (fasting): 178 mg/dl (normal range: 65–109 mg/dl) Creatinine: 1.0 mg/dl (normal range: 0.5–1.4 mg/dl) Blood urea nitrogen: 18 mg/dl (normal range: 7–30 mg/dl) Sodium: 141 mg/dl (normal range: 135–146 mg/dl) Potassium: 4.3 mg/dl (normal range: 3.5–5.3 mg/dl)
Lipid panel: Total cholesterol: 162 mg/dl (normal: <200 mg/dl) HDL cholesterol: 43 mg/dl (normal: ≥40 mg/dl) LDL cholesterol (calculated): 84 mg/dl (normal: <100 mg/dl)

Triglycerides: 177 mg/dl (normal: <150 mg/dl)
Cholesterol-to-HDL ratio: 3.8 (normal: <5.0)

AST: 14 IU/l (normal: 0–40 IU/l)
ALT: 19 IU/l (normal: 5–40 IU/l)
Alkaline phosphotase: 56 IU/l (normal: 35–125 IU/l)
A1C: 8.1% (normal: 4–6%)
Urine microalbumin: 45 mg (normal: <30 mg)
Assessment:
Uncontrolled type 2 diabetes (A1C >7%)
Obesity (BMI 32.4 kg/m2)
Hyperlipidemia (controlled with atorvastatin)
Peripheral neuropathy (distal and symmetrical by exam)
Hypertension (by previous chart data and exam)
Elevated urine microalbumin level
Self-care management/lifestyle deficits

Limited exercise
High carbohydrate intake
No SMBG program

Poor understanding of diabetes
(Source: http://spectrum.diabetesjournals.org/content/16/1/32.full)

Case 2: TB of the lung & uncontrolled DM 2:
A patient was admitted with a cough, fevers, night sweats and weight loss over several months. Her Tuberculin skin test was reactive and her chest x-ray revealed opacity in the right lung. A smear test was positive for acid-fast bacilli and culture and sensitivity results are pending. In addition, the patient had an uncontrolled DM episode which was caught late and she ended up with acute renal failure. She spent extra days in the hospital to take care her DM type2 complications.
She was discharged with diagnosis of tuberculosis of the lung to be treated as an outpatient. The culture test was eventually confirmed TB.

Case 3: A on C Renal Failure, Fracture, Heart Failure to CCU and stable DM 2:
A patient was admitted from a long term care facility. The elderly patient is 80 years old. She was admitted for renal failure. She was diagnosed with acute on chronic renal failure. She has a history of diabetes type 2. During this episode of care she developed an infection that resulted in respiratory failure and severe sepsis, Group A Streptococcus, pneumonia. She had an episode of fall which resulted in fracture of her left wrist and hair fracture of her left distal radius. As her health deteriorated, her attending physician also admitted her to the CCU for heart failure. Her health did not make a good candidate for surgery to fix her fracture of her wrist. Her left wrist was put in a cast. At discharge the patient was stable on diabetes medication and dialysis. She was discharged back to the long term care facility that she came from.

Case 4: Multi-location Cancer patient on Palliative:
A 79 year old patient visited ER for cold that had lasted a month. He was tested for bacterial infection and was negative. After further examination a skin lesion was noted, which was growing. The patient subsequently was admitted to inpatient ward and went for the biopsy of the

lesion which came back positive for melanoma. In addition, patient had prostate pain for a long time. After further examination of prostate, the PSA test, it was noted that patient required biopsy of the prostate. Patient went under a prostate biopsy, which showed prostate cancer. After further examination, the patient was designated ALC and was discharged to palliative care unit. He was discharged to a hospice after 12 days of ALC stay in the hospital. The patient was stable on palliative care at discharge.

Case 5: 1 cardiac with complications:
Patient was admitted for pain in his chest. He has had a history heart disease and high blood pressure. The ECG showed a ST elevation. Subsequently, patient underwent a PCI and the left main coronary artery dilation was performed. We used laser dilator with stent to perform dilation. Patient was stable after the intervention. He had DM 2 and needed immediate access to Reno-dialysis. The patient went through a dialysis treatment. With patient consent, we implemented a hemodialysis device subcutaneously for better care. Patient discharged in stable conditions.

Case 6: 1 ER to surgical, Fracture, re-admit category 7 days and some complication after:
Patient admitted to ER for fracture of the right arm. After an x-ray, it showed a complex fracture of right radius, and right ulna, and needed immediate intervention to fix the fracture. Subsequently, patient was admitted to surgery and underwent an open surgery for internal fixation of the fractures of both radius and ulna. The ulna required the use of staple and wire. All bleedings were cauterized. The patient was given antibiotics intraoperatively. The patient will need the removal of the implemented devices on a later date. Patient left the OR in stable conditions. Patient stayed overnight and was discharged next morning. The patient visited the ER 7 days later for fever, patient and redness in the area of the surgical sutures. The wound was cleaned and the cultures came back positive for E. Coli. The patient was admitted and was given antibiotics for 3 days. The patient was discharged in stable conditions.

Case 7: 1 Simple Day-Surg. with complication, so got admitted to Inpatient (Allergy to medication):
Patient was scheduled for a laparoscopic removal of gallbladder (Cholecystectomy). In addition, surgeon did exploration of the bile ducts and found several stones that were also removed. At postop, patient got a severe reaction to an antibiotic (penicillin) and she was admitted to inpatient for monitoring. During her stay, she developed fever and eventually went to anaphylactic shock and had 8 seizures. The patient was treated for the symptoms and was discharged in stable conditions.

Case 8: 1 cardiac with complications and Death:
90 year old patient was admitted from a nursing home. She had cardiac and renal failure. The patient was treated with ACE inhibitors and Digoxin. She was admitted to CCU, and for her renal failure she was put a dialysis device. Unfortunately, she developed a bacterial infection and had a severe reaction to the antibiotic medication which caused her to go into anaphylactic shock. She lost consciousness at 12 am and pronounced dead at 3:30 am.

Case 9: 1 Normal birth with postpartum hemorrhage complication:

An Anemic mother gave birth to a healthy newborn. However, after her normal birth, she developed increased blood loss and resulted in the postpartum hemorrhage. The surgeon was called in to do a partial hysterectomy. The patient undergone the hysterectomy and after a day of monitoring, she was discharged home in stable conditions.

Case 10: 1 HIV/AIDS patient treats for underlying factor (an infection):
A known HIV patient was admitted for treatment of his current infection. He was diagnosed with Pneumocystis carinii pneumonia and admitted for care. He was treated with antibiotic and he was discharged with stable conditions.

Case 11: Sore Throat Variants:
A 22-year old male college student presented with a two-day history of runny nose and slight cough, fever to 102 degrees Fahrenheit with occasional chills and muscle aches. He had taken two aspirin prior to seeing his physician. Physical examination was unremarkable accept for temperature of 101, marked pharyngitis, and clear rhinitis. RADT was positive for group A streptococci. The patient was given a prescription for Penicillin V 500 mg and told to take one pill twice a day for 10 days. Patient was also told to call the office if his symptoms did not improve in 3 days. He did not call to report any continued problems.

Case 12: Sore Throat Variants:
A 3-year old girl was seen in the hospital with a 24-hour history of acute throat and fever. On physical examination, her temperature was 103 Fahrenheit and her left tonsil acutely inflamed. The girl's mother told the physician that two of the child's siblings have recently had strep throat. An RADT was negative. A sample was taken for a throat culture and the patient's mother was given a prescription for Amoxicillin 250mg to give to the child three times per day for 10 days. The mother was told to call the office if the child's symptoms persisted. The throat culture report, filed in the patient's chart four days after the initial visit, was negative. The child was not seen again by the physician for 8 months at which time routine immunization were given.

Case 13: Sore Throat Variants:
A 42-year old man who works as a long-haul truck driver presented to the hospital with a stuffed nose, severe sore throat, and a cough. He had been perfectly well until 3 days ago. On physical examination, his temperature was 100 degrees Fahrenheit, his respiration rate was 18, and his lung had fine rales of the left side. The patient is a smoker. RADT was positive for group A Streptococci. Patient was given a prescription for Penicillin V 250 mg to take three times a day for 10 days. Patient was told to call the office if his symptoms did not improve in 3 days. He called two days later saying that his temperature had not come down and he still had a severe sore throat. The nurse practitioner told the patient to increase his dose of antibiotics to four times per day and she called in a prescription to the pharmacy for more penicillin V tabs to enable the patient to complete his 10-day course of antibiotic therapy. The patient did not contact the hospital again complaining of pharyngitis symptoms.

Case 14: Sore Throat Variants:
An 18-year old moderately retarded female who lives in a foster care home was brought to the hospital because she was complaining of feeling sick and having a mild sore throat. On physical examination, her temperature was 100 Fahrenheit, she had severe pharyngitis with a few

petechiae on her uvula, and erythematous tonsils, and cervical adenopathy. RADT was positive for group A streptococci. The patient's caregiver was given a prescription for 500 mg Amoxicillin (oral suspension) to give the girl twice a day for 10 days. The caregiver was told to call the office if the girl's symptoms persisted. The girl apparently improved, as the caregiver did not call the office.

Case 15: Sore Throat Variants:
A 5-year old boy was brought to the physician. His mother said he had been complaining of a sore throat and had exhibited a decreased appetite over the past 3-4 days. On physical examination, there was moderate pharyngitis with some exudates and marked bilateral swelling of the anterior cervical nodes. The patient's temperature was 99.8 degrees Fahrenheit. A throat culture was attempted but the patient resisted attempts to obtain an adequate specimen. The mother was given a prescription for Amoxicillin (oral suspension) 250mg to give to the child twice a day for 10 days. The mother was told to call the office if the child's symptoms persisted. The physician had no further contact with this patient or his mother.

Case 16: Sore Throat Variants:
A 52-year old female with a history of heart disease secondary to childhood rheumatic fever was seen by her physician following a 2-day history of several sore throat, fever, and myalgia. On physical examination, the woman's throat was found to be inflamed and anterior cervical adenopathy was present. Temperature was 101.2 degrees Fahrenheit. RADT was positive for group A streptococci. A throat culture was done. Patient was given a prescription for Erythromycin 500 mg to take three times a day for 10 days (she is allergic to penicillin). The patient was scheduled to return in 10days for a repeat throat culture. Upon return to the hospital, the patient's symptoms had improved. The repeat throat culture was negative.

Case 17: Sore Throat Variants:
A 42-year old female who works in a day care center was seen by her physician for a 2-day history of fever, malaise, and sore throat. On physical examination, she was found to have anterior cervical adenopathy and moderate pharyngitis. Her temperature was 100.2 degrees Fahrenheit. RADT was negative. A throat culture was done and patient was given a prescription for Penicillin V 250 mg to take three times a day. The culture report, which was filed in the patient's records 2 days after her hospital visit, was positive for group A streptococci. The patient was notified of the culture results. She indicated to the physician's nurse, who phoned with the results, that her symptoms were starting to subside and her temperature had returned to normal. The muse told the patient to continue taking the antibiotics for the full 10 days and to call the hospital if she did not improve. The patient returned to the hospital in six weeks with symptoms similar to her previous visit. The RADT was positive at that time. The patient indicated that several of the children in the day care center were taking Amoxicillin for strep throat infections. The physician discussed the need for good hand-washing techniques for the patient and for the children in the day care center. The patient was given a prescription for Penicillin V 250 mg to take three times a day for 10 days and told to contact the hospital if her symptoms did not improve or if she had a recurrent infection. The patient did not contact the hospital again for this problem.

Information from the Informatics Architecture team was composed of DAD dictionary and the selected data elements. Information on metadata and the frequencies of three core data elements (that is, Admin Source, Admin Type, and Encounter Type) from the BI Data Warehouse team will be the ADT system database and the core data elements it comprises. Information from Information Specialist and Clinical Information Support will be the metadata relationship between ADT and DAD at VIHA. Clinical reporting work with Cerner Person Management tools and Med2020 WinRec Abstracting on organizing of the metadata before it is stored in a data warehouse (refer to Figures 5 and 6). VIHA's privacy/security team was also interviewed to get a sense of data ownership and the necessary steps involved to get approval when using real data.

In the survey meetings, with VIHA personnel, it was verified with the core metadata of ADT/DAD at VIHA with questions scripted for the three main groups (Table 4). In this application of the script to the interviews, I've listed the main data elements and there were slight modifications done after consultation with the VIHA experts.

Table 4. Interview questions scripted with the three main groups involved in clinical reporting a VIHA.

Groups	Questions	Answers	Rationale	Application
Group 1 Architect	How can we format data that will make meaningful and closer to real data, in your opinion?	Make them into Data sets using CIHI's information for Island Health.	Focus on the current demographics and the information provided through CIHI's patient's trends of disease, hospitalization and readmission for BC and island health.	Can use standardized metadata of the data that CIHI requests from hospitals
Group 1 Architect	What is the list of the collected data in the island health?	CIHI and abstracted and coded records is collected	CIHI requests hospitals to submit data based on set data collection targets and standards	The majority of used collected data is from DAD and some ADT, therefore combining the 2 databases to form NoSQL database is representative
Group 1 Architect	How does ADT and DAD compare?	ADT is the system that doesn't collect diagnostic information, but DAD collects data based on all clinical data in the patient charts except that data collected in ADT	ADT doesn't use any coding schema to collect information. ADT is a system that aggregates the entered data for all the date and times from administration departments such as, admission.	ADT is location, medical service and inpatient to discharge, so we can add those columns while diagnosis and procedure are separate and can add those to the patient encounter even though they are separate

Group 1 Architect	What are the key associations in DAD (dx associated with the visit and that encounter)	Abstract identification, length of stay (LOS), patient demographics, admission data, discharge data, patient service information, service transfers, provider information, diagnosis information, intervention information	Requested by and regulated by CIHI	Associations can be based on the encounter and MRN ta hospital level with PHN as a primary key
Group 1 Architect	What are the key associations in ADT?	MRN, PHN, Name, DOB, DOD, address, family physicians	It is the most important system that holds the patient's non-clinical information	These are based at the patient encounter level and are represented by columns and rows in existing database for our NoSQL
Group 1 Architect	What is proper way to group DAD data and ADT data?	DAD data is grouped by diagnosis and interventions after discharge whereas ADT data is based on admission information	ADT is collected when patient is still in the hospital, but DAD data is recorded after patient leaves the healthcare facility	Therefore, combining ADT and DAD is already done at the hospital level and can have representation via non-relational database
Group 1 Architect	What are the data elements in DAD/ADT that we can derive robust queries from? (e.g., Dx, intervention codes, admission times?)	DAD: Intervention, admission location; dx code, provider service, location of admit; ADT: Encounter information, admission time and location and	DAD contain the clinical information that is collected ADT is the location, date and time of the visit and patient	Data elements for data are based on profiles at the metadata level and there is a data dictionary so we can simulate

		PHN	personal information	
Group 1 Architect	When looking for patient, what is more important (relating to queries) or is diagnosis more important or are the demographic, or prefixes and diagnosis types?	Diagnostic information is related to the MRN, PHN and encounter number and episode of care. The main patient's identifiers are the important IDs to find patient related clinical information	Patients are identified using their PHN, MRN and encounter number.	Encounter level queries are important as well as hospital level over patients, possible to represent encounters as rows in database
Group 2 Reporting	In terms of metadata, are the data profiles set up with dependencies in the system? We know that randomly generating the columns (there is a word document with the column list) is not correct since there are dependencies across the columns for specific rows. But how to mimic these dependencies? Can we group certain columns?	Randomly assigned admin date –need to calculate column while generating Transfer across location with encounter – acute care encounter – can change in level of care Surgical day care (admitted to and discharged from before ICU) Location (codes)- transfer and level of care Discharge patient -> long term care -> partition to AR days? ALC versus LTC Dependencies Reporters report the	Produce standard reports hourly, daily, weekly, monthly, yearly with no errors	For reporting, the metadata is supposed to be standardized at the enterprise architecture. Dependencies in the data can be simulated with the correct metadata

		encounter as flat file? We need to ask data warehouse team how the data is put into sequence. It is the metadata as DAD as per the data warehouse – may be the Spotlight program with the data profiles show dependencies. Diagnosis – most reasonable diagnosis, multiple diagnosis Co-morbidities à put in column Dependencies on diagnosis → clinical dependencies (difficult to setup algorithm) -complexity indicator of diagnosis could be used		
Group 2 Reporting	What is the integration between ADT and DAD for potential queries?	ADT – from Cerner Ops BI Team – DAD and ADT ADT – source of truth Medical Services (check the numbers) – -affiliation (hospitalist versus fam physician versus specialist) – CPIS #, and it is a unique category number	ADT is implemented from vendor and source of truth and automated, DAD is abstraction and utilizes source therefore the 2 databases are already linked	Combining ADT and DAD is possible and representative of hospital system while supporting clinical reporting and benchmarking our simulations

		ADT does not have diagnosis – it is related to diagnosis Acute and outside of acute care (ADT) -treat as source -including ambulatory -DAD and services (attached to ADT, encounter ID) -encounter ID would be a primary		
Group 2 Reporting	What are the important metadata from the columns for the DAD and ADT databases?	Diagnosis, encounter, intervention	Significant relevance to reporting to CIHI	Able to show similar queries in simulation
Group 2 Reporting	What columns (for DAD and ADT) would be frequently utilized for queries and which ones not used often?	Diagnosis, encounter, intervention	Standardized reporting	Able to show similar queries in simulation
Group 2 Reporting	VIHA uses MRN as its local primary key or health care number. The PHN (provincial number) for that patient could be several or would it be identified uniquely? (Are MRN, Health Care Number and PHN composite primary keys?) In what way would the metadata be linked to MRN versus PHN?	Composite keys MRN and FIN for an encounter. I've confirmed that I have access to the Cerner data model tool and can view all composite keys.	Primary keys are important for data integrity and no errors while linking encounter to patient	Keys need to be representative in the constructed database

Group 2 Reporting	Encounters with FINs and MRNs are unique for an active encounter that has a number of dependencies like accession number, inpatient location, diagnosis, and so on. However, our database is generated using one large file distributed via a filing system into a NoSQL database; therefore, we would use the following to identify the patient per visit to allow for queries to drill down to an event? What is the integration with encounters to the patient visit?	Encounter ID is important. The encounter alias pools are also important.	Encounter level data important to standard reporting and data integrity	Simulation patient encounters at hospital level to represent clinical reporting
Group 2 Reporting	To form queries in a NoSQL database, we need to develop a key store or indices that group or form families of columns with primary keys part of the grouping. The following are some queries that we tested using a few family groupings or a group schema. Any suggestions to forming "family groups" in the metadata? For instance, here is some script with a Row Key and groupings that we tested: HBase org.apache.Hadoop.HBase.MapReduce.ImportTsv '-	In the Cerner Data model there are groupings in the table (children and parent)	Key stores important to index data because foundation of system is based on patient encounter	Need to utilize technologies to create key stores and unique indexes of the encounters in order to query the data

	Dimporttsv.separator=,' - Dimporttsv.columns=HBASE_ROW_KEY, encounter:MRN, FIN, Encounter_Number, Admit_Type, Admit_Source, Admit_Time, Discharge_Date, Discharge_Time, Medical Service, Discharge_Disposition, MRP,\ Demographics:MRN, PHN, Name, DOB, Address, Family_Physician, DOD,\ Transfer:Transfer_Date, Transfer_time, Location_building, Location_unit, Location_room, Location_time ADT_data /user/dchrimes/data/ADT/ex1data.csv			
Group 2 Reporting	In these queries (or just one of them), what suggestions or important groupings are there of necessary columns or rows to retrieve accurate results? Does diagnosis included several columns as its overall metadata and/or query?	Multiple diagnoses might need a complexity column associated…	Important queries: Frequency of Diagnosis (Dx) Code with LOS Frequency of Diagnosis (Dx) Code with LOS Diagnosis Code with Discharge date and	Need to incorporate the important queries with cases or scenarios of patient encounters as proof of concept

			Discharge time Diagnosis Code with Unit Transfer Occurrence Diagnosis Code with Location building, Location Unit, Location Room, Location Bed, Discharge Disposition Diagnosis Code with Encounter Type and LOS Diagnosis Code with Medical Services and LOS Highest LOS for MRNs with Admit date. Frequency (or number) of Admit category with Discharge_Date Provider Service with Diagnosis codes	

Group 2 Reporting	Calculations involved with the DAD are difficult to emulate in our large NoSQL database other than simply adding an additional column look correct to you based on how DAD/ADT dates and times are calculated? How does the system calculate the Length of Stay (LOS) from Admission time/date and discharge time/date? Would these calculations be fairly accurate?	Reporters report the encounter as flat file? We need to ask data warehouse team how the data is put into sequence. Calculations are important for the groupings b\and we might need to put in columns to represent.	[Discharge time/date] – [Admission time/date] = length of stay (LOS) [Current date] – [Birth date] = Age [Left ED date/time] – [Admission to ED date/time] = Wait time in ED Intervention start date/time = needs to be between [Admission time/date] and [Discharge time/date] (Intervention) Episode Duration = Should be less than length of stay (LOS) Transfer In/Out Date = Should be between [Admission time/date] and [Discharge time/date] Days in Unit = Should be less than or equal to	Combining the columns we need to be able to perform these basic calculations

			length of stay (LOS)	
Group 3 Data-Warehouse	Dependencies/ Constraints in database: For reporting and extracting purposes is there a script that is run? Or does Med2020 or WinRec incorporate the dependencies? Is there a quick reference to dependencies among data profiles for lookup on the reporting side of DAD or would this be with the BI team?	ADT table needs to be fixed with transfer with encounter. Medical Services disconnected from encounter, it changes over time. In transfer (outpatient to inpatient) – ER to major surgery could be admitted, transfer date/time (bed transfer history) Location bed – inpatient – is a metric, admitted thru Emerg (physician admits patient the time of admitted starts…) ALC – involved with transfers	Dependencies and constraints are important for archiving data to query later on and combine with or among other data sets	Similar to key stores we need dependencies in our database to be representative of existing system
Group 3 Data-Warehouse	Dependencies/ Constraints in database: How are dependencies (or row constraints) established in the data warehouse? Is there a shell script run? Is there a quick reference available for lookup?	The arrival date and time involved when the patient goes thru admission. We probably don't need to include the time thru admission and could simply this by removing some columns. Discharge disposition – involves death and there	Certain data elements with standardized metadata are necessary for the data to be accurate	Need to generate same metadata with accurate dependencies

		are 3 options, one is went home, second is acute care facility and third is death.		
Group 3 Data-Warehouse	DAD and ADT Integration: How are the DAD and ADT integrated? Like encounter_ID is primary key with composites of MRN and PHN? What about medical services in ADT are that crossed over to the DAD from ADT? Or is the linkage that of DAD and Services attached to the ADT?	+Transfer is related to encounter +VIHA has continuous encounter, reclassify encounter +Nuances -> considered inpatient -> 2 encounters, 1 visit +Mother.baby scenario – inpatient -> outpatient event or outpatient à inpatient event	Integration is not necessary for system to work but only to query the data ad hoc or correctly, currently no real time or streaming data	Integration depends on patient healthcare numbers from system at each encounter and linkage between ADT and DAD via indexed rows
Group 3 Data-Warehouse	DAD and ADT Integration: What is Medical Service in ADT? How is multiple diagnoses incorporated?	DAD – is one continuous encounter, triage solution in entering notes and codes -Abstraction has diagnosis à most responsible physician (not commonly used with encounter… for queries) -superfluous or bogus diagnosis (all events have a diagnosis including bogus -no intervention tied to encounter…but is with	Medical Services is not currently utilized in clinical reporting because it is not DAD abstracted but could be utilized in data warehouse. The reason is due to CIHI's data standards	Can integrate medical services and other metadata from ADT with direct linkage to metadata from DAD

		physician and nursing unit, orthopedics… Intervention – can't associate with what provider and who did what Associate transfers -> could be combination of medical services and intervention -expensive drugs / antibiotics (not cat scans or mri's) Don't group from MRPs but specialists and hospitalists Scenario for Heart Surgery -heart surgery, followed by another heart problem followed by another (in the hospital) Diagnosis type: pre-existing (not primary) and post existing (acquired infection in hospital) Primary diagnosis plus Type 6 Scenario: HIV -> proxy is pneumonia MRP – population health Proxy – transfers and		

		activities to query -diabetes & ulcers is manifestation -> diabetic ulcer, non-diabetic		
Group 3 Data-Warehouse	DAD and ADT Integration: How are transfers or movement of patients from one bed to another incorporated?	Transfer units –separate from transfer in ADT -regular and special care unit (ICU-ICCU) There is no association at surgery with anesthesiologists -surgical episode is not associated -surgical episode, prefix -more than one anesthesiologists and more than one specialists cannot track the activities under complex interactions among health professionals with surgery, for instance	Transfers are important to ADT and flow of patients in the system as their encounters progress and change	Can use transfers and locations in the database as simulated metadata of known profiles from hospital
Group 3 Data-Warehouse	Trends and Queries: Frequencies in the DAD/ADT? Are there common trends from DAD in Island Health that should we aware of?	Case Mixed groupings, RIWs Case Mixed Groupings are rolled up in clinical category Comorbidities is a machine process and part of the groupings	Abstraction of the data into group is important for hospital systems because data can be diverse and also to check financial constraints and billing quickly and accurately	Combining columns against encounter rows is already implemented at the hospital level therefore ADT and DAD combination is relevant and simulation valuable

Group 3 Data-Warehouse	Trends and Queries: Are there groupings of data fields in SQL or other queries available as a reference?	RIWs used for stats Calculations by specifics via DAD reporting team -interpretation	Groupings important to query quickly and accurately	Groupings allows building and construct of database to add columns progressively based on the encounter
Group 3 Data-Warehouse	Trends and Queries: What queries are important and are up-coming?	In terms of groupings, transfer could be aggregates of cohorts involved… like for specific diagnosis (as the business case in a clinical perspective…) Diagnosis is in DAD only so business case groups with Diagnosis and then combines with ADT to group by Age…	Diagnosis is important because it is health outcome of hospital. Groupings important as performance metrics	Simulating queries based on encounters for diagnosis is therefore primary to NoSQL database

At VIHA, health informatics architecture has direct relation to the DAD abstracting, as it is a manual process, and dependent on *Admit Type* and *Admit Source* obtained from Cerner's hospital system. The emergency system is separate from the ADT, and there are also planned procedures in the triage that are not part of the ADT system. Doctors refer patients to beds and for mental health there are specific instances of patients moving to different beds if they are deemed a danger to themselves or others. Doctors and Nurses refer to the patient encounter as the source of "truth" of patient encounters in the ADT system. Each patient can have multiple encounter numbers and administrative errors can occur as a result of the system to avoid administrative errors and complex encounters. With over a million encounters annually at VIHA this is important. In contrast, DAD is an attribute of the encounter, mostly diagnosis and discharge, while ADT represents a person's relationship to the hospital system with medical services and patient location(s). The encounter can drastically change over the course of the patient visit and several scenarios were discussed over the interview related to simulated data (Table 5). Outpatient and inpatient encounters are different and separate, and this study inpatient encounters were only used/formulated. However, this study did include patient movement in hospital (which is currently not queried at large levels) and patient transfers. A transfer is a change in the encounter that is not always represented by digital documentation; for example, a patient may be transferred to Nanaimo Regional General Hospital (NRGH) in Nanaimo, and then receive a new encounter after being discharged from RJH. Therefore, it is difficult to track the patient flow among the different patient transfers and different hospitals.

The data construction framework was used by this study to extract the appropriate data profiles from the data dictionaries and data standard definitions for the ADT system and DAD abstract manual (CIHI, 2012). The data used to test performance was circumvented with the interviews with different VIHA stakeholders, several requirements and scenarios, and 17 clinical cases are outlined in Table 5 that were identified for a realistic proof-of-concept BDA and data visualization: the interface should be interactive, simple, ANSI-SQL or SQL-like, and visualization enabled. Moreover, the platform should be able to offer the necessary tools to implement new and advanced analytics and act as a recommendation and/or an expert system when utilized; this is required, as clinicians are interested in answering specific scenarios of inpatient encounters with accurate data.

Table 5. Use cases and patient encounter scenarios related to metadata of the patient visit and it placement in the database related to query output.

Case #	Case	Metadata	Database Build	Query Output
1	Uncontrolled Type 2 diabetes & Complex comorbidities	ICD10-CA, MRN, PHN and LOS, Discharge	DAD with Diagnosis Codes, patient IDs and Discharge in Columns	ICD10-CA codes with counts, frequencies or max values for patient encounters
2	TB of the lung & uncontrolled DM 2	ICD10-CA, MRN, PHN, Inpatient Encounter, Location, Unit Transfer	DAD and ADT columns	ICD10-CA and encounter type codes with counts, frequencies or max values for patient encounters
3	A on C Renal Failure, Fracture, Heart Failure to CCU and stable DM 2	ICD10-CA, MRN, PHN, Intervention (CCI), Episode, Unit Transfer, Bed Location, CCU codes, Discharge	DAD and ADT columns	ICD10-CA, CCI and encounter types and unit transfer and bed location codes with counts, frequencies or max values for patient encounters
4	Multi-location Cancer patient on Palliative	ICD10-CA, MRN, PHN, Intervention (CCI), Surgery, Episode, Bed Location, Transfer to ALC Unit, Medical Services and Patient Services, Discharge	DAD and ADT columns	ICD10-CA, CCI and encounter types and unit transfer and bed location and medical service codes with counts, frequencies or max values for patient encounters
5	1 cardiac with complications	ICD10-CA, MRN, PHN, Intervention (CCI), Surgery, Episode, Bed Location, Transfer, Medical Services, Discharge	DAD and ADT columns	ICD10-CA, CCI and encounter types and transfer codes with counts, frequencies or max values for patient encounters

6	1 ER to surgical, Fracture, re-admit category 7 days and some complication after	ICD10-CA, MRN, PHN, Intervention (CCI), Surgery, Episode, Bed Location, Medical Services, Progress Notes, Discharge, Re-Admission	DAD and ADT Columns	ICD10-CA, CCI and medical services and re-admit codes with counts, frequencies or max values for patient encounters
7	1 Simple Day-Surg. with complication, so got admitted to Inpatient (Allergy to medication)	ICD10-CA, MRN, PHN, Intervention (CCI), Surgery, Bed Location, Medical Services, Discharge	DAD and ADT Columns	ICD10-CA, CCI and medical services codes with counts, frequencies or max values for patient encounters
8	1 cardiac with complications and Death	ICD10-CA, MRN, PHN, Intervention (CCI), Episode, Bed Location, Transfer, Medical Services, Discharge Disposition	DAD and ADT Columns	ICD10-CA, CCI and medical services, discharge disposition and transfer codes with counts, frequencies or max values for patient encounters
9	1 Normal birth with postpartum hemorrhage complication	ICD10-CA, MRN, PHN, Intervention (CCI), Surgery, Episode, Bed Location, Medical Services, Discharge	DAD and ADT Columns	ICD10-CA, CCI and medical services and discharge codes with counts, frequencies or max values for patient encounters
10	1 HIV/AIDS patient treats for underlying factor (an infection)	ICD10-CA, MRN, PHN, Medical Services, Discharge	DAD and ADT Columns	ICD10-CA, and medical services codes with counts, frequencies or max values for patient encounters
11	Strep A infection	ICD10-CA, MRN, PHN, Medical Services, Discharge	DAD and ADT Columns	ICD10-CA, and medical services codes with counts, frequencies or max values for patient

				encounters
12	Cold but Negative Strep A. Child with throat culture	ICD10-CA, MRN, PHN, Medical Services, Discharge	DAD and ADT Columns	ICD10-CA, and medical services codes with counts, frequencies or max values for patient encounters
13	Adult patient with Strep A. positive and physical exam	ICD10-CA, MRN, PHN, Medical Services, Patient Services, Discharge	DAD and ADT Columns	ICD10-CA, patient and medical services codes with counts, frequencies or max values for patient encounters
14	Severe Pharyngitis with physical exam	ICD10-CA, MRN, PHN, Medical Services, Patient Services, Discharge	DAD and ADT Columns	ICD10-CA, patient and medical services codes with counts, frequencies or max values for patient encounters
15	Child, moderate Pharyngitis, throat culture negative, physical exam	ICD10-CA, MRN, PHN, Medical Services, Discharge	DAD and ADT Columns	ICD10-CA, and medical services codes with counts, frequencies or max values for patient encounters
16	Adult, history of heart disease, Positive culture for Strep A.	ICD10-CA, MRN, PHN, Medical Services, Patient Services, Discharge	DAD and ADT Columns	ICD10-CA, patient and medical services codes with counts, frequencies or max values for patient encounters
17	Adult, physical exam, moderate pharyngitis, positive for strep A. culture and positive second time, re-admit	ICD10-CA, MRN, PHN, Medical Services, Patient Services, Discharge	DAD and ADT Columns	ICD10-CA, patient and medical services codes with counts, frequencies or max values for patient, readmit encounters

ii. Initial Technical Generation

Metadata was set over 90 columns and randomized based on data dictionary examples and from VIHA interviews. For example, metadata for the diagnostic column was set with standardized metadata of International Classification of Disease version 10 Canadian or ICD-10-CA codes, and Personal Health Number (PHN) has ten numerical digits while the patient's Medical Record Number (MRN) for that encounter has nine numerical digits (Table 6). During the interviews and confirmation of the necessary columns to generate the emulation of aggregated data, all data elements and their required fields, as well as primary and dependent keys, were recorded. This was applied to a data generator in Excel (Table 6) before generating over the Oracle 11g Express via PL/SQL to establish the column names and created metadata using constrained data profiles for each randomized row of dummy patient data via PL/SQL code in Oracle Express (Table 7). The generator included all important data profiles and dependencies were established through primary keys over selected columns. The data profiles and metadata established by the interviews were maintained through scripts that randomly generated data to form an initial one million records. Additionally, current reporting limitations of the different combinations of the DAD and ADT data were recorded, to form accurate emulation of the patient data and simulation of the existing and future modified queries. Therefore, the randomized data simulated was a representation of current production.

The data warehouse team working with health professionals for clinical reporting can rely on comma-separated value (*.csv*) formats when importing and exporting data. Therefore, this study opted to use the ingested *.csv* files directly for analytics instead of HBase, which had previously been used on this platform along with Apache Phoenix and its SQL-like code (e.g., White, 2015). Three data sizes (50 million, one and three billion records) were used as benchmark checks of how different packages (Apache Spark and Drill) scaled with data size.

Table 6. Excel example of generated metadata examples to for the NoSQL database.

DAD - Column Name	DAD- Data Type	DAD - Data Examples (Dictionary)
Reporting_Prov NUMBER(5),	Number (5)	90202,91202, 902036, 90204,90206, 91206, 90217, 91217, 90501, 91501, 90502, 91502, 90507, 90508, 91508, 90510, 90511, 90851, 91851, 90854
MRN VARCHAR2(11),	Varchar2(11)	ex: J123456789,
Admission_Date VARCHAR2(8),	Date(YYYYMMDD)	99991231
Admission_Time VARCHAR2(5),	Time(hh:mm)	00:00 - 23:59
Discharge_Date VARCHAR2(8),	Date(YYYYMMDD)	99991231
Discharge_Time VARCHAR2(5),	Time(hh:mm)	00:00 - 23:59
LOS NUMBER(4),	Number (4)	9999
Last_Name VARCHAR2(10),	Text (10)	von hanstiens
First_Name VARCHAR2(10),	Text (10)	Jordan

Middle_Name VARCHAR2(10),	Text (10)	Maryam
Gender VARCHAR2(1),	Char(1)	F,M, O, U
Birth_Date VARCHAR2(8),	Date(YYYYMMDD)	99991231
Postal_Code VARCHAR2(7),	Char(7)	V8M 16T, XX, 99
Residence_Code NUMBER(5),	Number(5)	00001-00013, blank
Responsibility_for_payment NUMBER(2),	Number(2)	01-08, blank
HCN_Prov VARCHAR2(2),	Varchar2(2)	BC,AB,MB,NB,NL,NS,NT,NU,ON,PE,QC,SK,YT, 99, CA, blank
Health_Care_Number VARCHAR2(12),	Varchar2(12)	1, 0, 8, 9, 9123456789
Admit_Category VARCHAR2(1),	Char(1)	L, N, R, S, U
Admit_by_Ambulance VARCHAR2(1),	Char(1)	A, C, G, N
Institute_From NUMBER(5),	Number (5)	0,1,3,4,6,7,8, 9006, 9007, 90973, 90974, 98107, 98875, 99996 (different institutions)
Readmit_Code NUMBER(1),	Number (1)	1-5, 9
Entry_Code VARCHAR2(1),	Char(1)	C, D, E, N, P, S
Arrival_Date_in_ER VARCHAR2(8),	Date(YYYYMMDD)	99991231
Arrival_Time_in_ER VARCHAR2(5),	Time(hh:mm)	23:59
Date_Patient_Left_ED VARCHAR2(8),	Date(YYYYMMDD)	99991231
Time_Patient_Left_ED VARCHAR2(5),	Time(hh:mm)	23:59
Time_Pt_left_ED_unknown VARCHAR2(1),	Char(1)	Y
Wait_Time_in_ED VARCHAR2(5),	Time(hh:mm)	23:59
Discharge_Disposition VARCHAR2(2),	Varchar2(2)	01,02,03,04,05,06,07,08,09,12
Institution_To NUMBER(5),	Number(5)	0,1,3,4,6,7,8,90006,90007,90973,90974,98107,98975,99996 (different institutions)
Discharge_Site VARCHAR2(1),	Char(1)	B,C,J,K,L,M,N,P,Q,R,S,T,V,W
Most_Responsible_Site VARCHAR2(1),	Char(1)	B,C,J,K,L,M,N,P,Q,R,S,T,V,W
Diagnosis_Type VARCHAR2(1),	Varchar2(1)	M,1,2,3,5,9,0,W,X,Y
Diagnosis_Prefix VARCHAR2(1),	Varchar2(1)	I, P, Q, 5, 6, 8
Diagnosis_Code VARCHAR2(7),	Varchar2(7)	K22.6, N39.0, D50.0, Z51.3, Y83.1, E10.10, I11, I50.0, N18.9, J13, J18.1, J44.0, J18.9, K52.9, M17.5, S32.000, W10, U98.0
Diagnosis_Cluster VARCHAR2(1),	Char(1)	A to Y
Glasgo_Coma_Scale VARCHAR2(2),	Varchar2(2)	03-15, 99

Interven_Occurrence NUMBER(1),	Number(1)	1,2,3 ….
Interven_Episode_Start_Date VARCHAR2(8),	Date(YYYYMMDD)	99991231
Interven_Code VARCHAR2(12),	Varchar2(12)	1.CL.54.LA-LM, 1.CL.55.LA.FE, 1.CL.89.NP-LP, 1.CL.89.NP-LM, 1.CS.59.JA-AD, 1.CE.56.JA, 1.CU.59.LA-AD
Interven_Attribute_Status VARCHAR2(1),	Varchar2(1)	Y, N
Interven_Attribute_Location VARCHAR2(1),	varchar2(1)	B,L,R,S
Interven_Attribute_Extent VARCHAR2(1),	varchar2(1)	3, At, AV, U, VE
Interven_Options VARCHAR2(1),	varchar2(1)	x, A,I, P, T, B, F,R, T, Y, Z
Interven_Provider_Number NUMBER(5),	Number(5)	99999, 59999, 88888
Interven_provider_service NUMBER(5),	Number(5)	00010, 00001, 00013, 00020
Interven_Start_Time_unknown VARCHAR2(1),	char(1)	Y, blank
Episode_Duration NUMBER(4),	number(4)	1 to 4320
Interven_Preadmit_Flag VARCHAR2(1),	varchar(1)	Y, blank
Anesthestic_Grade VARCHAR2(1),	varchar2(01)	01,02,03,04,05,09
Anesthetistic_Technique NUMBER(1),	number(1)	0 to 9
Anesthetist_ID VARCHAR2(5),	varchar2(5)	00001-99999
Interven_Location VARCHAR2(2),	varchar2(2)	01 to 11
Out_of_Hospital_ooh_indicator VARCHAR2(1),	char(1)	Y, blank
Out_of_hospital_ooh_number NUMBER(5),	number(5)	98501, 99999, 97501
Unplanned_return_to_OR VARCHAR2(1),	char(1)	Y, blank
Provider_Occurrence NUMBER(1),	Number(1)	1,2,3 ….
Provider_Number NUMBER(5),	Number(5)	00001-99999
Provider_Type VARCHAR2(1),	Char(1)	M,2,3,4,5,7,8,9,H,P,W,X,Y
Provider_Service VARCHAR2(5),	varchar2(5)	00000-00130
PHN NUMBER(9),	number(9)	912345678
Patient_Service_Occurrence NUMBER(1),	Number(1)	1,2,3…
Patient_Service NUMBER(3),	Number (3)	99,76,58,64,10,12,20,30,33,34,39,40,51,52,53, 531,541,722
Patient_Service_Type VARCHAR2(1),	Char(1)	M,W,X,Y

Transfer_In_Date VARCHAR2(8),	Date(YYYYMMDD)	99991231
Transfer_Out_Date VARCHAR2(8),	Date(YYYYMMDD)	99991231
Patient_Service_Days NUMBER(3),	Varchar(5)	00001 - 99999, blank
Service_Nursing_Area VARCHAR2(10),	Varchar2(10)	ALC, Palliative, Rehab, psych
Unit_Transfer_Occurrence NUMBER(1),	Number(1)	1,2,3 ….
Transfer_Nursing_Unit VARCHAR2(10),	Varchar2(10)	ALC, Palliative, Rehab, psych
Date_of_Transfer_In VARCHAR2(8),	Date(YYYYMMDD)	99991231
Transfer_Hours VARCHAR2(5),	Time(hh:mm)	23:59
Days_in_Unit VARCHAR2(5)	varchar2(5)	99999

ADT- Column Name	ADT- Data Type	ADT- Data Examples (Dictionary)
MRN	10 character, including site prefix (ex. RJH) = prefix=J	ex: J123456789,
FIN	8 digits	87456321
Encounter Number	10 character	1234567891
Admit Type	alpha	D,E,N,S,P,C
Admit Source	characters	Hospital, clinic, police, ambulance
Admit Time	4 digits, using 24 hour clock. Midnight is 00:00 hours	00:00 - 23:59
Admit Date	YYYYMMDD	99991231
Discharge time	5 digits, using 24 hour clock. Midnight is 00:00 hours	23:59
Discharge date	YYYYMMDD	99991231
Encounter Type	10 Characters	Surg, daysurg, ortho, ped, Gen, Inpatient, outpatient
medical services	25 characters	IV, ICU, CCU, Lab, RAD, Farm
Discharge Disposition	Identifies the location where the patient was discharged or the status of the patient on discharge.	01,02,03,04,05,06,07,08,09,12
Most Responsible Provider	25 characters	John Hopson
PHN	9 digits	987456321
Last Name	Can contain hyphen, spaces, etc.	von hanstiens
First Name	Can contain hyphen, spaces, etc.	Jordan
Middle Name	Can contain hyphen, spaces, etc.	Maryam
DOB	YYYYMMDD	99991231
Address	25 characters	234 straight road, Victoria
Family Physician	25 characters	John Hopson
DOD	YYYYMMDD	99991231

DOD time	hh:mm	23:59
Transfer date	YYYYMMDD	99991231
Transfer time	hh:mm	23:59
Location building	4 characters	RJH, VGH, SJGH
Location unit	4 characters	PCC, ALC, DT, PO, PrO
Location room	5 characters	S800, N900, S400, N500
location bed	1 characters	A, B, 1, 2, 3, 4

Table 7. PL/SQL programming code for data generator via Oracle Express.

```
** PL/SQL to create the DAD table */
CREATE  TABLE DAD (
        Reporting_Prov   NUMBER(5),
        MRN   VARCHAR2(11),
        Admission_Date   VARCHAR2(10),
        Admission_Time   VARCHAR2(5),
        Discharge_Date   VARCHAR2(10),
        Discharge_Time   VARCHAR2(5),
        LOS  NUMBER(4),
        Last_Name    VARCHAR2(10),
        First_Name   VARCHAR2(10),
        Middle_Name  VARCHAR2(10),
        Gender  VARCHAR2(1),
        Birth_Date    VARCHAR2(10),
        Postal_Code   VARCHAR2(7),
        Residence_Code  NUMBER(5),
        Responsibility_for_payment  NUMBER(2),
        HCN_Prov   VARCHAR2(2),
        Health_Care_Number   VARCHAR2(12),
        Admit_Category   VARCHAR2(1),
        Admit_by_Ambulance   VARCHAR2(1),
        Institute_From  NUMBER(5),
        Readmit_Code  NUMBER(1),
        Entry_Code   VARCHAR2(1),
        Arrival_Date_in_ER  VARCHAR2(10),
        Arrival_Time_in_ER  VARCHAR2(5),
        Date_Patient_Left_ED  VARCHAR2(10),
        Time_Patient_Left_ED  VARCHAR2(5),
        Time_Pt_left_ED_unknown   VARCHAR2(1),
        Wait_Time_in_ED  VARCHAR2(5),
        Discharge_Disposition   VARCHAR2(2),
        Institution_To  NUMBER(5),
        Discharge_Site   VARCHAR2(1),
        Most_Responsible_Site   VARCHAR2(1),
        Diagnosis_Occurrence  NUMBER(1),
        Diagnosis_Type   VARCHAR2(1),
```

Diagnosis_Prefix VARCHAR2(1),
Diagnosis_Code VARCHAR2(7),
Diagnosis_Cluster VARCHAR2(1),
Glasgo_Coma_Scale VARCHAR2(2),
Interven_Occurrence NUMBER(1),
Interven_Episode_Start_Date VARCHAR2(10),
Interven_Code VARCHAR2(13),
Interven_Attribute_Status VARCHAR2(1),
Interven_Attribute_Location VARCHAR2(1),
Interven_Attribute_Extent VARCHAR2(1),
Interven_Options VARCHAR2(1),
Interven_Provider_Number NUMBER(5),
Interven_provider_service NUMBER(5),
Interven_Start_Time_unknown VARCHAR2(1),
Episode_Duration NUMBER(4),
Interven_Preadmit_Flag VARCHAR2(1),
Anesthestic_Grade VARCHAR2(1),
Anesthetistic_Technique NUMBER(1),
Anesthetist_ID VARCHAR2(5),
Interven_Location VARCHAR2(2),
Out_of_Hospital_ooh_indicator VARCHAR2(1),
Out_of_hospital_ooh_number NUMBER(5),
Unplanned_return_to_OR VARCHAR2(1),
Provider_Occurrence NUMBER(1),
Provider_Number NUMBER(5),
Provider_Type VARCHAR2(1),
Provider_Service VARCHAR2(5),
PHN NUMBER(9),
Patient_Service_Occurrence NUMBER(1),
Patient_Service NUMBER(3),
Patient_Service_Type VARCHAR2(1),
Transfer_In_Date VARCHAR2(10),
Transfer_Out_Date VARCHAR2(10),
Patient_Service_Days NUMBER(3),
Service_Nursing_Area VARCHAR2(10),
Unit_Transfer_Occurrence NUMBER(1),
Transfer_Nursing_Unit VARCHAR2(10),
Date_of_Transfer_In VARCHAR2(10),
Transfer_Hours VARCHAR2(5),
Days_in_Unit VARCHAR2(5)
FIN NUMBER NUMBER (9),
Encounter_Number NUMBER(10)
Admit_Type VARCHAR2(10),
Admit_Date VARCHAR2(10),
Admit_Time VARCHAR2(5),
Encounter_Type VARCHAR2(10),

```
        Medical_Services VARCHAR(25),
        Most_Responsible_Provider VARCHAR(25),
        DOD VARCHAR(10),
        Location_Building VARCHAR2(4),
        Location_Unit VARCHAR2(4),
        Location_Room VARCHAR2(4),
        Location_Bed VARCHAR2(1),
);
```

/*PL/ SQL to insert rows to DAD table */

```
DECLARE
  MaxN NUMBER(10):= 1000; /*Number of rows are generated */
  n    NUMBER(10);
  r    NUMBER(2);
  v1   NUMBER(5);
  v2   VARCHAR2(11);
  v3   VARCHAR2(10);
  v4   VARCHAR2(5);
  v5   VARCHAR2(10);
  v6   VARCHAR2(5);
  v7   NUMBER(4);
  v8   VARCHAR2(10);
  v9   VARCHAR2(10);
  v10  VARCHAR2(10);
  v11  VARCHAR2(1);
  v12  VARCHAR2(10);
  v13  VARCHAR2(7);
  v14  NUMBER(5);
  v15  NUMBER(2);
  v16  VARCHAR2(2);
  v17  VARCHAR2(12);
  v18  VARCHAR2(1);
  v19  VARCHAR2(1);
  v20  NUMBER(5);
  v21  NUMBER(1);
  v22  VARCHAR2(1);
  v23  VARCHAR2(10);
  v24  VARCHAR2(5);
  v25  VARCHAR2(10);
  v26  VARCHAR2(5);
  v27  VARCHAR2(1);
  v28  VARCHAR2(5);
  v29  VARCHAR2(2);
  v30  NUMBER(5);
  v31  VARCHAR2(1);
  v32  VARCHAR2(1);
```

```
v33  NUMBER(1);
v34  VARCHAR2(1);
v35  VARCHAR2(1);
v36  VARCHAR2(7);
v37  VARCHAR2(1);
v38  VARCHAR2(2);
v39  NUMBER(1);
v40  VARCHAR2(10);
v41  VARCHAR2(12);
v42  VARCHAR2(1);
v43  VARCHAR2(1);
v44  VARCHAR2(1);
v45  VARCHAR2(1);
v46  NUMBER(5);
v47  NUMBER(5);
v48  VARCHAR2(1);
v49  NUMBER(4);
v50  VARCHAR2(1);
v51  VARCHAR2(1);
v52  NUMBER(1);
v53  VARCHAR2(5);
v54  VARCHAR2(2);
v55  VARCHAR2(1);
v56  NUMBER(5);
v57  VARCHAR2(1);
v58  NUMBER(1);
v59  NUMBER(5);
v60  VARCHAR2(1);
v61  VARCHAR2(5);
v62  NUMBER(9);
v63  NUMBER(1);
v64  NUMBER(3);
v65  VARCHAR2(1);
v66  VARCHAR2(10);
v67  VARCHAR2(10);
v68  NUMBER(3);
v69  VARCHAR2(10);
v70  NUMBER(1);
v71  VARCHAR2(10);
v72  VARCHAR2(10);
v73  VARCHAR2(5);
v74  VARCHAR2(5);
v75 NUMBER (9),
v76 NUMBER(10)
v77 VARCHAR2(10)
v78 VARCHAR2(10),
```

```
  v79 VARCHAR2(5),
  v80 VARCHAR2(10),
  v81 VARCHAR(25),
  v82 VARCHAR(25),
  v83 DOD VARCHAR(10),
  v84 VARCHAR2(4),
  v85 VARCHAR2(4),
  v86 VARCHAR2(4),
  v87 VARCHAR2(1),


BEGIN
  FOR n IN 1..MaxN LOOP
    v1 := TRUNC(DBMS_RANDOM.VALUE(90001,99999));
    v2 := 'J' || TO_CHAR(DBMS_RANDOM.VALUE(100000001,999999999),'999999999');
    v3 :=
TO_CHAR(DBMS_RANDOM.VALUE(1990,2014),'9999')||LTRIM(TO_CHAR(DBMS_RAN
DOM.VALUE(1,12),'00'))||LTRIM(TO_CHAR(DBMS_RANDOM.VALUE(1,31),'00'));
    v4 :=
LTRIM(TO_CHAR(DBMS_RANDOM.VALUE(0,23),'00'))||':'||LTRIM(TO_CHAR(DBMS_R
ANDOM.VALUE(0,59),'00'));
    v5 :=
TO_CHAR(DBMS_RANDOM.VALUE(1990,2014),'9999')||LTRIM(TO_CHAR(DBMS_RAN
DOM.VALUE(1,12),'00'))||LTRIM(TO_CHAR(DBMS_RANDOM.VALUE(1,31),'00'));
    v6 :=
LTRIM(TO_CHAR(DBMS_RANDOM.VALUE(0,23),'00'))||':'||LTRIM(TO_CHAR(DBMS_R
ANDOM.VALUE(0,59),'00'));
    v7 := TRUNC(DBMS_RANDOM.VALUE(1,999));
    v8 := 'L'||TO_CHAR(n);
    v9 := 'F'||TO_CHAR(n);
    v10 := 'M'||TO_CHAR(n);
    IF (TRUNC(DBMS_RANDOM.VALUE(0,2))=1) THEN v11 := 'M';
      ELSE v11 := 'F';
    END IF;
    v12 :=
TO_CHAR(DBMS_RANDOM.VALUE(1920,2010),'9999')||LTRIM(TO_CHAR(DBMS_RAN
DOM.VALUE(1,12),'00'))||LTRIM(TO_CHAR(DBMS_RANDOM.VALUE(1,31),'00'));
    v13 := 'V8N 6G7';
    v14 := TRUNC(DBMS_RANDOM.VALUE(1,99999));
    v15 := TRUNC(DBMS_RANDOM.VALUE(1,8));
    v16 := DBMS_RANDOM.string('U',2);
    v17 := DBMS_RANDOM.string('U',12);
    v18 := DBMS_RANDOM.string('U',1);
    v19 := DBMS_RANDOM.string('U',1);
    v20 := TRUNC(DBMS_RANDOM.VALUE(90001,99999));
    v21 := TRUNC(DBMS_RANDOM.VALUE(1,9));
```

```
   v22 := DBMS_RANDOM.string('U',1);
   v23 :=
TO_CHAR(DBMS_RANDOM.VALUE(1990,2014),'9999')||LTRIM(TO_CHAR(DBMS_RAN
DOM.VALUE(1,12),'00'))||LTRIM(TO_CHAR(DBMS_RANDOM.VALUE(1,31),'00'));
   v24 :=
LTRIM(TO_CHAR(DBMS_RANDOM.VALUE(0,23),'00'))||':'||LTRIM(TO_CHAR(DBMS_R
ANDOM.VALUE(0,59),'00'));
   v25 := v3;
   v26 :=
LTRIM(TO_CHAR(DBMS_RANDOM.VALUE(0,23),'00'))||':'||LTRIM(TO_CHAR(DBMS_R
ANDOM.VALUE(0,59),'00'));
   v27 := 'Y';
   v28 :=
LTRIM(TO_CHAR(DBMS_RANDOM.VALUE(0,23),'00'))||':'||LTRIM(TO_CHAR(DBMS_R
ANDOM.VALUE(0,59),'00'));
   v29 := LTRIM(TO_CHAR(DBMS_RANDOM.VALUE(1,12),'00'));
   v30 := TRUNC(DBMS_RANDOM.VALUE(90001,99999));
   v31 := DBMS_RANDOM.string('U',1);
   v32 := DBMS_RANDOM.string('U',1);
   v33 := TRUNC(DBMS_RANDOM.VALUE(1,9));
   v34 := DBMS_RANDOM.string('X',1);
   v35 := DBMS_RANDOM.string('X',1);
   v36 :=
DBMS_RANDOM.string('U',1)||LTRIM(TO_CHAR(DBMS_RANDOM.VALUE(0,99),'00'))||'.'|
|TO_CHAR(TRUNC(DBMS_RANDOM.VALUE(1,999)));
   v37 := DBMS_RANDOM.string('U',1);
   v38 := LTRIM(TO_CHAR(DBMS_RANDOM.VALUE(3,15),'00'));
   v39 := TRUNC(DBMS_RANDOM.VALUE(0,9));
   v40 :=
TO_CHAR(DBMS_RANDOM.VALUE(1990,2014),'9999')||LTRIM(TO_CHAR(DBMS_RAN
DOM.VALUE(1,12),'00'))||LTRIM(TO_CHAR(DBMS_RANDOM.VALUE(1,31),'00'));
   v41 :=
'1.'||DBMS_RANDOM.string('U',2)||LTRIM(TO_CHAR(DBMS_RANDOM.VALUE(0,99),'00')
)||'.'||DBMS_RANDOM.string('U',2)||'-'||DBMS_RANDOM.string('U',2);
   IF (TRUNC(DBMS_RANDOM.VALUE(0,2))=1) THEN v42 := 'Y';
     ELSE v42 := 'N';
   END IF;
   v43 := DBMS_RANDOM.string('U',1);
   v44 := DBMS_RANDOM.string('X',1);
   v45 := DBMS_RANDOM.string('A',1);
   v46 := TRUNC(DBMS_RANDOM.VALUE(10001,99999));
   v47 := TRUNC(DBMS_RANDOM.VALUE(1,99999));
   v48 := '-';
   v49 := TRUNC(DBMS_RANDOM.VALUE(1,4230));
   v50 := 'Y';
   v51 := DBMS_RANDOM.string('U',1);
```

```
   v52 := TRUNC(DBMS_RANDOM.VALUE(0,9));
   v53 := TRUNC(DBMS_RANDOM.VALUE(1,99999));
   v54 := TRUNC(DBMS_RANDOM.VALUE(1,11));
   v55 := 'Y';
   v56 := TRUNC(DBMS_RANDOM.VALUE(10000,99999));
   v57 := NULL;
   v58 := TRUNC(DBMS_RANDOM.VALUE(1,9));
   v59 := TRUNC(DBMS_RANDOM.VALUE(1,99999));
   v60 := DBMS_RANDOM.string('X',1);
   v61 := LTRIM(TO_CHAR(DBMS_RANDOM.VALUE(0,99999),'00000'));
   v62 := TRUNC(DBMS_RANDOM.VALUE(100000000,999999999));
   v63 := TRUNC(DBMS_RANDOM.VALUE(1,9));
   v64 := TRUNC(DBMS_RANDOM.VALUE(10,999));
   v65 := DBMS_RANDOM.string('U',1);
   v66 :=
TO_CHAR(DBMS_RANDOM.VALUE(1990,2014),'9999')||LTRIM(TO_CHAR(DBMS_RAN
DOM.VALUE(1,12),'00'))||LTRIM(TO_CHAR(DBMS_RANDOM.VALUE(1,31),'00'));
   v67 :=
TO_CHAR(DBMS_RANDOM.VALUE(1970,2014),'9999')||LTRIM(TO_CHAR(DBMS_RAN
DOM.VALUE(1,12),'00'))||LTRIM(TO_CHAR(DBMS_RANDOM.VALUE(1,31),'00'));
   v68 := TRUNC(DBMS_RANDOM.VALUE(1,999));
   v69 := DBMS_RANDOM.string('A',10);
   v70 := TRUNC(DBMS_RANDOM.VALUE(1,9));
   v71 := DBMS_RANDOM.string('A',10);
   v72 :=
TO_CHAR(DBMS_RANDOM.VALUE(1990,2014),'9999')||LTRIM(TO_CHAR(DBMS_RAN
DOM.VALUE(1,12),'00'))||LTRIM(TO_CHAR(DBMS_RANDOM.VALUE(1,31),'00'));
   v73 :=
LTRIM(TO_CHAR(DBMS_RANDOM.VALUE(0,23),'00'))||':'||LTRIM(TO_CHAR(DBMS_R
ANDOM.VALUE(0,59),'00'));
   v74 := TRUNC(DBMS_RANDOM.VALUE(1,999));

   v75 := TRUNC(DBMS_RANDOM.VALUE(1,999999999));
   v76 := TRUNC(DBMS_RANDOM.VALUE(1,9999999999));
   v77 := DBMS_RANDOM.string('U',1);
   v78 := DBMS_RANDOM.string('U',1);
   v79 := DBMS_RANDOM.string('U',1);
   v80 :=
TO_CHAR(DBMS_RANDOM.VALUE(1990,2014),'9999')||LTRIM(TO_CHAR(DBMS_RAN
DOM.VALUE(1,12),'00'))||LTRIM(TO_CHAR(DBMS_RANDOM.VALUE(1,31),'00'));
   v81 :=
LTRIM(TO_CHAR(DBMS_RANDOM.VALUE(0,23),'00'))||':'||LTRIM(TO_CHAR(DBMS_R
ANDOM.VALUE(0,59),'00'));
   v82 := DBMS_RANDOM.string('X',1);
```

```
   v83 :=
TO_CHAR(DBMS_RANDOM.VALUE(1920,2010),'9999')||LTRIM(TO_CHAR(DBMS_RAN
DOM.VALUE(1,12),'00'))||LTRIM(TO_CHAR(DBMS_RANDOM.VALUE(1,31),'00'));
   v84 :=
   v85 := DBMS_RANDOM.string('A',10);
   v86 := DBMS_RANDOM.string('A',10);
   v87 := DBMS_RANDOM.string('A',10);

   INSERT INTO DAD VALUES (v1, v2, v3, v4, v5, v6, v7, v8, v9, v10,
                                    v11, v12, v13, v14, v15, v16, v17, v18, v19, v20,
                                    v21, v22, v23, v24, v25, v26, v27, v28, v29, v30,
                                    v31, v32, v33, v34, v35, v36, v37, v38, v39, v40,
                                    v41, v42, v43, v44, v45, v46, v47, v48, v49, v50,
                                    v51, v52, v53, v54, v55, v56, v57, v58, v59, v60,
                                    v61, v62, v63, v64, v65, v66, v67, v68, v69, v70,
                                    v71, v72, v73, v74, v75, v76, v77, v78, v79, v80,
                                    v81, v82, v83, v84, v85, v86, v87);
 END LOOP;
END;
```

iii. Establishing the Analytical Process

It is important to note that this study is about performance testing of ADT/DAD queries of a distributed filing system (Apache-Hadoop) with a processing (MapReduce) configuration on an emulated NoSQL database (HBase) of patient data. The platforms tested the totally randomized generated data with duplicates for every 50 million patient's encounters with that of replicated groupings, frequencies, dependencies, etc. in the queries. On top of these three data sizes the analytics were applied as a pipelined data processing framework for the BDA platform in accordance with review of Health Big Data requirements (Kuo et al., 2015; Chrimes et al., 2016a; Chrimes et al., 2016b). The process included five stages or phases that coincided with the challenges outlined in Section 1.0 and corroborative the study's objectives:

1) Data acquisition

a. Data emulation and modeling

Over the span of 12 months in 2014, several interviews were conducted with business intelligence, data warehouse, clinical reporting, application platform, and health informatics architecture teams employed at VIHA. During these interviews, a patient data system generated from hospital admissions (based on encounter types) and a discharge system (based on diagnoses and procedures) were established. Furthermore, data profiles, including dependencies and the importance of the metadata for the reporting performance of the hospital, were confirmed and verified for use in over the platform (using HBase and Hadoop). All workflows involved with the teams to query data at the hospital level were also recorded.

In the emulated dataset, each row represented encounter-based patient data, with diagnoses, interventions, and procedures specific to that patient, that the current ADT system has in its database schema linked to a bigger data warehouse, which includes DAD (and Tables 3-5 show

the clinical cases). This patient-specific structure in the database allowed for active updates for accurate patient querying over the platform, simulated throughout the lifetime of that person.

All necessary data fields were populated for one million records before replication to form one billion records. The recorded workflow provided a guideline to form the NoSQL database as a large distributed flat file. It was further emulated the patient-specific rows across the columns according to the existing abstraction; HBase established a wide range of indexes for each unique row, and each row contained a key value that was linked to the family of qualifiers and primary keys (columns). The HBase operations were specific to family qualifiers at each iteration; therefore, the data was patient-centric combined with certain DAD data (from different sources of metadata) in the rows and columns, such that summary of diagnosis or medical services as a whole could be queried.

Chawla and Davis (2013) showed that utilization of ICD diagnosis codes over a patient-centered framework allowed for a seamless integration with a variety of data from electronic healthcare systems with patient-centric ADT; this method could accurately query readmission rates and quality of care ratings, and demonstrate meaningful use and any impact on personal and population health. Therefore, the platform used a similar methodology to establish the structure of the data model of combining encounter-based ADT with standardized diagnosis; every encounter has a separate diagnosis, procedures, and most responsible provider.

b. Data translation

The beauty of a big data ecosystem is that it has a wide range of tools and technologies for bulkloading and accessing large datasets from different sources. *Sqoop,* for example, can be used to ease the transfer between existing relational databases and a BDA platform (Xu et al., 2016). For collecting and gathering unstructured data, such as logs, one can use Flume. Since the performance tests of queries on the platform relied on data emulation, it was used, as a proof-of-concept, the usual high-speed file transfer technologies (such as SCP and GridFTP) to transfer data to the HPC parallel file system (GPFS). It was then used the Hadoop and HBase as NoSQL database bulkload utilities to ingest the data.

After the metadata, used in the emulated database, was verified from the existing relational database, it was established as a patient-centric SQL database using Phoenix on top of the non-relational NoSQL HBase store. To establish data structure, the EncounterID was set as a big data integer (so that it can reach billions of integers without limitation) and indexed based on that integer via HBase for each unique at each and every iteration that followed. This indexed-value column, unique for every row, causes MapReduce to sort the KV stores for every one of the iterations that can increase the integrity of the data and increase its secured access once distributed. Once distributed, this allowed for SQL-like scenarios to be used in the evaluation of the performance of the BDA platform and accuracy of the emulated patient database.

2) Data maintenance

The emulated data was stored and maintained in the HPC parallel File (~500GB) and in over the BDA platform under HDFS. The replication factor for HDFS was set to three for fault tolerance. The large volume of data sets was re-utilized to test the performance of different use cases or queries conducted by the analytics platform. This required innovation, in a team setting, to develop stages in the methodology unique to BDA configurations related to healthcare databases, in order for the database to maintain data integrity.

3) Data integration

The data was transformed into appropriate formats for subsequent and sequential analysis of the analytical tools depending on the study question and analytic algorithm used. This step was very important because the SQL-like Phoenix queries had to produce the same results as the current production system at VIHA. All results were tested under a specific data size and comparable time for analysis, whether the query was simple or complex. The data results also had to show the exact same columns after the SQL-like queries over the constraint of the family qualifiers (as primary keys). Over a series of tests, certain columns were included or excluded as qualifiers in the SQL code for constraints. Once the results were accurate and the same as those benchmarked, those qualifiers remained for each of the iterations run via Hadoop, to generate the one billion totals. Since the results from the queries were known and the data was integrated accurately with DAD and ADT for each row as an encounter, of the simulations of the queries could be compared for accuracy and performance.

4) Data analysis

In this step, the study conducted a proof-of-concept analysis of task-related use cases specific to clinical reporting. The queries were evaluated based on the performance and accuracy of the BDA framework over the one billion rows. For example, a task-based scenario for the analysis included:

a. Analysis questions/scenarios

A typical analysis scenario was: Clinicians suspect that frequent movement of patients within the hospital can worsen outcomes. This is especially true in those who are prone to confusion due to changes in their environment (i.e., the elderly). Furthermore, frequent room changes can exacerbate confusion (this applies only to hospital systems recording information that is not utilized in the overall data warehouse). If this occurs, a physician may attribute the confusion to the possible onset of an infection, a disease, or a mental illness, resulting in unnecessary laboratory tests and medications. Moreover, changes in the locations or movement of the patients may also lead new healthcare workers or the hospital system to detect subtle changes in a patient's condition or mental baseline, as a pro-active measure to avoid sepsis or episodic circumstances from occurring.

b. Analytic algorithms and developing tools

To handle intensive computation, simplified algorithms were applied and distributed over database nodes over the parallel supercomputing architecture. For instance, there was some default MapReduce-based *apriori* data-mining algorithm to find associated patterns in the dataset. The customized MapReduce templates were tailored to be used via Phoenix (later, in a separate part of this study, it was also tested similar algorithms via Apache Spark and Apache Drill) on the HBase database nodes. To give the structured data the same semantics as its relational database counterpart, HBase was, therefore, selected to use Apache Phoenix (SQL-like) on HBase because Phoenix is a relational database layer that runs on top of HBase and offers a low-latency SQL-like access to HBase by running multiple parallel HBase scans on different RegionServers (White, 2015). For developing some of the software pipeline, the plan was to establish and engineer alternative products with Spark such as Jupyter and Zeppelin (in a separate study) to work over Hadoop and establish a query GUI interface to interactively run all test queries (as in this study) simultaneously and display all durations to generate results. Apache Drill was also selected because the same queries tested in Phoenix and Spark can be used plus its interface can be integrated over Hadoop.

c. Pattern validation and presentation

The study undertook more than five phases of the configuration process (over several months) to query the data distributed. The initial aim of assessing how well the models will perform against a large dataset was first carried out with publicly available annual (2005-2006) inventory of pharmaceuticals (~5 MB). Once the pharmaceutical data ingested on the Hadoop/MapReduce framework showed the same results benchmarked as from MS Excel pivot tables then the simulated data was ingested and queried at much larger volumes with additional configurations. Simulated queried-results from the platform were to follow the DAD reporting for health outcomes at the hospital level and each row was deemed to represent one patient encounter. For this to succeed, domain experts and physicians were involved in the validation process and interpretation of the results and end users' usability of the query tools. Since the data was randomized at one million records and replicated iteratively at 50 million to one billion then to three billion the data results were known; therefore, the trends detected will be randomized clusters only. If other patterns arise, the BDA platform did not accurately query the data and further configurations will be needed to have accurate results before one and three billion are queried.

5) Data privacy protection

Ensuring patient data privacy is an important requirement in this study. Existing data privacy protection studies can be categorized into two main approaches: privacy-by-regulations and privacy-by-techniques. The privacy-by-regulations approach assumes that policies and regulations are generally enforceable and protect user data from accidental disclosure or misuse while facilitating informed choice options. These regulations are important, as a broader implementation of BDA, but the privacy-by-techniques approach is more related to this project because innovative software engineering steps could be formed or monitored to adhere to the appropriate policies or regulations (e.g., steps taken to allow encrypting of HBase indexes in public KV stores (Pattuk et al., 2014; Chawla & Davis, 2013). Research in this field is aimed at developing privacy methods and systems with provable privacy guarantees (Erdmann, 2009; Win et al., 2006; Benaloh, Chase, Horvitz, & Lauter, 2009). Among these methods, data masking was

seen as the same as generating a data replication of the data via encrypted distribution most effectively conceals the subject's identity while preserving the information, relationship, and context (Chawla & Davis, 2013; Jin, Ahn, & Hu 2011). Nevertheless, there are challenges in using data masking via generating replication, including questions about what defines sensitive data, how to replicate and mask it successfully, and how it is referenced or configured to provide the necessary privacy while allowing access to the data in the search algorithms (Pencarrick Hertzman, Meagher, & McGrail, 2013).

The established framework of the platform used WestGrid's existing security and the privacy of its supercomputing platform while reviewing and identifying regulations for eventually using real patient data over the platform (external to the hospital's data warehouse). This method included four steps:

Step 1. The main purpose of this step involved identifying and cataloging sensitive data across the health authority. Typically, this is carried out by business/security analysts based on regulations by the Province. The goal of which is to generate a comprehensive list of sensitive elements specific to the organization and discover the associated tables, columns, and relationships across the data warehouse (Moselle, 2015). HBase creates indexes for each row of data that cannot be queried with direct access, and queries can only be generated when accessing the deployment manager (DM) on the platform. That is, the data cannot be viewed at all by anyone at any time or for any duration; only queries can show the data that is HBase-specific and non-recognizable without Hadoop and HBase running as well as the correct scripts to view it. This is supported by Pattuk et al. (2013) verified in their platform that the HBase index process encrypts its key stores such that the risk of security breach is removed.

Dates and addresses are known to be sensitive data (Spivey, 2014); however, these elements were not masked in this study, as it only had emulated data. For real data, the testing and validation plan, in addition to the rest of the security measures taken, was to use both HBase encryption and compression to further protect the data. The HBase transparent encryption and compression was an attractive feature for securing access to healthcare data with high performance processing (Nishimura, Das, Agrawal, & Abbadi, 2012; Nguyen, Wynden, & Sun, 2011; Sun, 2013; Taylor, 2010).

Step 2. Executing data replication, as a generator over the platform, worked in conjunction with business/security analysts to identify the masking or encrypt-required algorithms that represented optimal techniques to replace the original sensitive data. Once the data was replicated to form an HBase dataset distributed across the DataNodes to form a database, it could only be queried via Apache Phoenix or an HBase client.

Step 3. Test of the replicated dataset was executed an application process to test whether the resulting masked data could be modified to view. A real dataset (large annual inventory of pharmaceuticals) was tested and verified firstly. Since studies have shown that the distribution of data using Hadoop has many inherent processes that restrict access to running ingestions (Chang et al. 2014; Karim et al., 2013; Mohammed et al., 2014; Sakr & Elgammal, 2016; Taylor, 2010), it was assumed that no one has access and, therefore, no one can modify the data during replication or masking. The replication routines were refined by continually checking the data for any inaccuracies once queries were performed and results were displayed. The replication process must be accurately indexed with family of primary keys in the patient-centric data. Consequently, data was removed and deleted when inaccuracies were found, then the replication and query performance tests were re-executed.

Step 4. Review was carried with the related regulations in regards to privacy protection regulations and principles, such as the HIPPAA, Freedom of Information and Protection of Privacy Act (FIPPA), Personal Information Protection Act (PIPA), and BC privacy protection principles for data acquisition. Reviewing these regulations is important in perspective of regional, provincial, and federal laws governing the use of personal health data that BDA tools need to overcome to produce a well-engineered database search (e.g., Canada Health Infoway, 2012; Moselle, 2015). Review of data management policies at VIHA and public disclosure policies regarding real patient data utilization required by provincial government approvals (e.g., Moselle, 2015; Pencarrick Hertzman et al., 2013) aided specifically in validating the use of the BDA platform and ascertaining any potential repercussions of the process, including access to public KV stores established in semi-permanent databases of HBase distributed by Hadoop.

3.5 Implementing Framework – Hadoop-HBase-Phoenix

In this section, the steps and experiences implementing the technical framework and application of a BDA platform are described. The established BDA platform will be used to benchmark the performance of end users' querying of current and future reporting of VIHA's clinical data warehouse (i.e., in production, spans more than 50 years of circa 14 TB). To accomplish this, Hadoop environment (including the Hadoop HDFS) from a source was installed and configured on the WestGrid cluster, and a dynamic Hadoop job was launched. The Hadoop (version 2.6.0) was configured by hdfs-site.xml and a MapReduce frame (Mohammed et al., 2014), configured via mapred-site.xml, that was run under the Hadoop resource manager Yarn (with configuration file yarn-site.xml). The number of replicas was set to three. To interact with HDFS, command scripts were run to automate the ingestion step (generate data replication in the exact format specified by SQL script to the nodes).

The BDA platform was built on top of the available open-source software (HBase). HBase (NoSQL version 0.98.11) is a NoSQL database composed of the main deployment master (DM) and fail-over master, the RegionServers holding HBASE data, and ZooKeeper, which contained services to allocate data locality (ZooKeeper, 2016), of three nodes, that orchestrated that ensemble. The xml configuration files were *HBase-site.xml* and the *HBase-env.sh* were adjusted to improve the performance of HBase. HBase was chosen due to its NoSQL services and many other features, especially linear and modular scalability. In addition, it allows for SQL-like layers via Apache Phoenix to be configured on top of HBase big-tables.

The construction and build of the framework with HBase (NoSQL) and Hadoop (HDFS) established the BDA platform. This construct coincided with and is enforced by the existing architecture of the WestGrid clusters at UVic (secure login via LDAP directory service accounts to deployment database nodes and restricted accounts to dedicated nodes). It was initially running the architecture of the platform with five worker nodes and one master node (each with 12 cores), and planned on increasing the (dedicated) nodes to 11 and possibly to 101, as well as incorporating a non-dedicated set of virtual machines on WestGrid's openstack cloud. The data were distributed in parallel on the nodes via a balanced allocation to each local disk with running part of the batch jobs in a serial computing process. Figure 7 shows the framework of the HBase/Hadoop dynamic cluster on the WestGrid cluster, called Hermes.

Deployment of the Hadoop environment on the nodes was carried out via a sequence of setup scripts that the user calls after loading the necessary modules. These setup scripts create an initial configuration depending on the number of nodes chosen when launching the job. The user can adjust those configurations to match the needs of the job and its performance. Currently, with six nodes, one master node is deployed and the rest act as slaves or workers. The services that the master node and slave nodes run are shown in Figure 7.

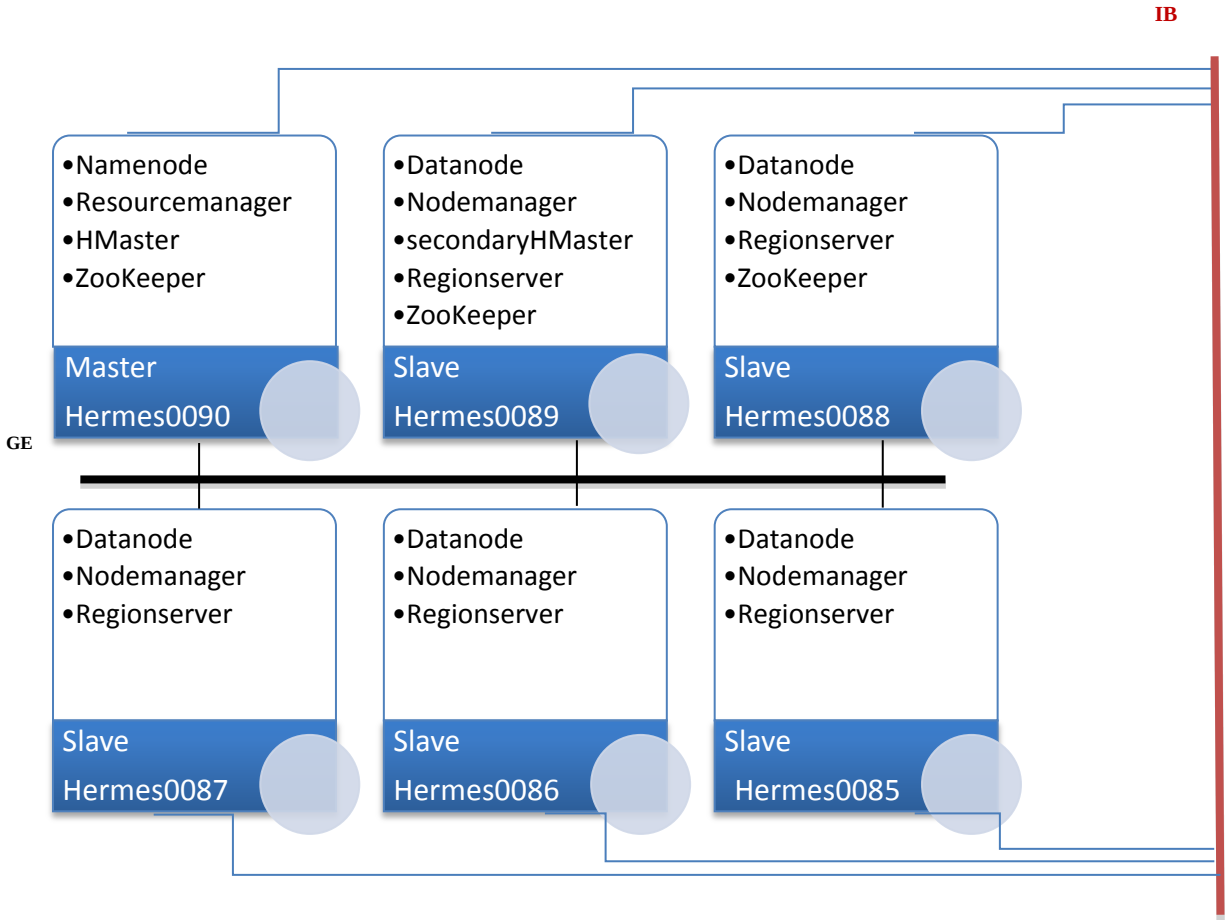

Figure 7. Construction of HBase NoSQL database with dynamic Hadoop cluster and master and slave/worker services at WestGrid Supercomputing. Hermes is the database nodes; GE and IB represent the kind of network connectivity between the nodes: Gig Ethernet (GE), usually used for the management

network, and the InfiniBand (IB), used to provide low-latency high-bandwidth (~40GB/s) communications between the nodes.

Making the DBA platform InfiniBand-enabled was challenging, as most of the Hadoop environment services rely on the hostname to get the IP address of the machine. Since the hostnames on a cluster are usually assigned to their management network, the setup scripts and the configuration files required adjustment. The InfiniBand was used because it offers low latency (in us) and high bandwidth (~40Gb/s) connectivity between the nodes. Yarn, Hadoop's resource and job manager, unfortunately still partly used the Gig-Ethernet interface when orchestrating between the nodes, but the data transfer was carried out on the InfiniBand.

The queries via Apache Phoenix (version 4.3.0) resided as a thin SQL-like layer on HBase. This allowed ingested data to form structured schema-based data in the NoSQL database. Phoenix can run SQL-like queries against the HBase data. Similar to the HBase shell, Phoenix is equipped with a python interface to run SQL statements and it utilizes a *.csv* file bulkloader tool to ingest a large flat file using MapReduce. The load balancing between the RegionServers (e.g., "salt bucket" script parameter) was set to the number of slaves or worker nodes that allowed ingested data to be balanced and distributed evenly.

The pathway to running ingestions and queries from the build of the BDA platform on the existing HPC was as follows: *.csv* flat files generated → Module Load → HDFS ingestion(s) → Bulkloads/HBase → Apache Phoenix Queries. Under this sequence, the Apache Phoenix module loaded after Hadoop and HBase SQL code was directed and then iteratively run to ingest 50 million rows to the existing NoSQL HBase database (see 4.0 Results for details). Additionally, the derived queries were compared with clinical cases and how that interacted with the performance of the platform that was representative of the clinical reporting at VIHA (refer to 4.0 Results section for more detail) Table on Query Description. This pathway was tested in iteration up to three billion records (once generated) for comparison of the combination of HBase-Phoenix versus Phoenix-Spark or an Apache Spark Plugin (Apache Phoenix, 2016).

Performance was measured with three main processes: HDFS ingestion(s), bulkloads to HBase, and query times via Phoenix. Three flat files in *.csv* format were ingested to HDFS. One contained 10 records and was used for quick testing, and two others were used for the actual benchmarking: one with 50 million records and the other with one and three billion records. One measurement ingestion time in total for iterations and overall was established to achieve the total desired number of records (i.e., one billion and three billion from 50 million replicated). A calculation was also set for T_i (N) to be the time to ingest N records or rows, which was calculated as ingestion efficiency (IE) for each *i* iteration with total records achieved (Chrimes et al., 2016a; Chrimes et al., 2016b), as follows:

$$IE = \frac{1(3)\text{B} \times Ti(50\text{M})}{50\text{M} \times Ti(1(3)\text{B}}$$

This was the same calculation for simplified calculation for query efficiency (QE):

$$QE = \frac{1(3)\text{B} \times Ti(50\text{M})}{50\text{M} \times Ti(1(3)\text{B}}$$

where M is million records, B is billion records, and *T* is cumulative time for the iterations (*i*). There were three separate runs at 50 million, one billion, and three billion, respectively.

Measurements were recorded of stop-watched times displayed on screen in the SQL-python line of queries for different query types derived from the performance of the BDA platform. The durations included the times to retrieve patient-related results of queries. The python interface with Apache Phoenix was configured to display the query times with HBase/Hadoop running in the background. These values were used for 22 queries in total: two simple queries with wildcard column selection, ten simple queries that did not involve more than three columns in the primary keys, and ten complex queries that had >3 columns selected (Chrimes et al., 2016a). These queries were chosen based on surveys and interviews with VIHA experts on clinical reporting. The same separation of the exact same simple versus complex queries was also derived to later compare with Apache Spark and Apache Drill action filters, transformations, and queries.

3.6 Implementing Framework – Hadoop-Spark-Drill

A. Overview

Apache Spark (version 1.3.0) was also built from source and installed to use on HBase and the Hadoop cluster. The intent was to compare different query tools like Apache Spark and Drill, implemented in over the BDA platform, against Apache Phoenix using similar SQL-like queries. The entire software stack used in the platform has at its bottom HDFS (Figure 8). HDFS is known to scale and perform well in the data space (over distributed files over multiple nodes). Yarn was the resource manager of Hadoop and services of scheduling incongruent to running the Hadoop jobs. In addition to the MapReduce component, Yarn and HDFS constitute the main components (Taylor, 2010).

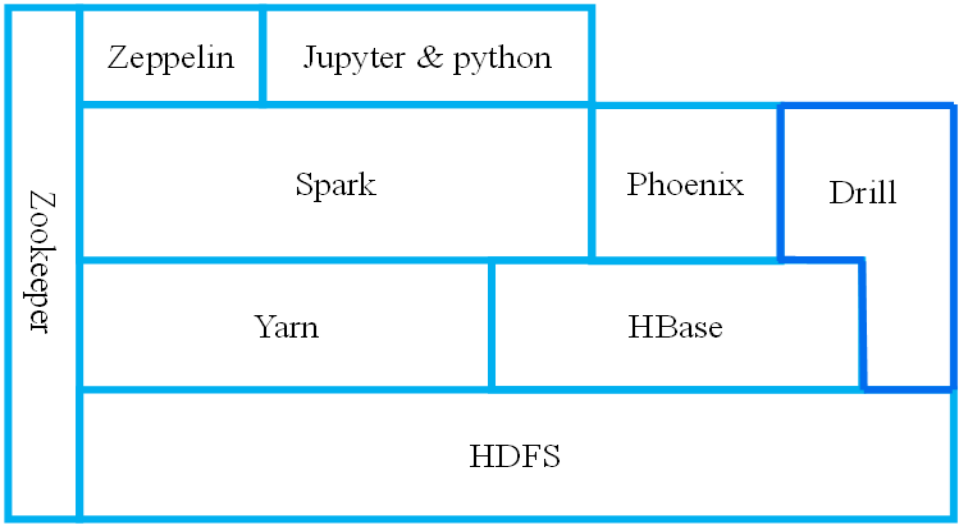

Figure 8. Healthcare Big Data Analytics (HBDA) software stacks in our study. ZooKeeper is a resource allocator, Yarn is a resource manager, HDFS is Hadoop Distributed Filing System, HBase is a NoSQL database, Phoenix is Apache Phoenix (query tool on HBase), Spark is Apache Spark (query tool with specialized transformation with Yarn), Drill is Apache Drill (query tool with specialized configuration to Hadoop with ZooKeeper), and Zeppelin and Jupyter are interfaces on local host web clients using Hadoop ingestion and a Spark module.

The bulk of the framework was comprised of open-source packages of Hadoop's ecosystem. Even though several configurations were done to the Hadoop ecosystem to optimize running on the WestGrid dedicated cluster, no hardware modification was needed; possible future changes could be made to meet minimum recommended RAM, disk space, etc. requirements per node (e.g., see Cloudera's guidelines 2016). Hadoop provides the robust, fault-tolerant HDFS inspired by Google's file system (Chang et al., 2006), as well as a Java-based API that allows parallel processing across the nodes of the cluster using the MapReduce paradigm. The platform was used in Python with Java to run jobs and ingestions. Hadoop comes with Job and Task Trackers that keep track of the programs' execution across the nodes of the cluster. These Job and Task Trackers are important for Hadoop to work on a platform in unison with MapReduce and other ingestion steps involved with HBase, ZooKeeper, Spark, and Drill. There have been many contributors, both academic and commercial (Yahoo being the largest), to using Hadoop over a BDA platform, and a broad and rapidly growing user community (Henschen, 2010).

In the platform different versions of Hadoop ecosystem was built, configured, and tested. All versions of Hadoop were managed via modules and environment packages. The following module command lists all the available Hadoop versions:

```
$module avail Hadoop
Hadoop/1.2.1(default) Hadoop/2.2.0 Hadoop/2.6.0 Hadoop/2.6.2
```

For this study, Hadoop 2.6.2 was used with replication set to three in all over each of the runs. Instead of using MapReduce as the main service for the algorithmic space, Apache Spark was chosen, as it covers larger algorithmic and tool spaces and outperforms Hadoop's MapReduce to HBase as bulkload.

B. Spark

Currently, Apache Spark, its action filters and transformations and resilient distributed dataset (RDD) format, is at the heart of the processing queries over the platform aligned with Hadoop (Sitto & Presser, 2015). It was covered a substantial data space because it could read data from HDFS and HBase. Its speed can be fast as a result of its judicious use of memory to cache the data, because Spark's main operators of transformations to form actions/filters were applied to an immutable RDD (Karau et al., 2015; Sitto & Presser, 2015). Each transformation produces an RDD that needs to be cached in memory and/or persisted to disk, depending on the user's choice (Dhyani & Barthwal, 2014; Karau et al., 2015). In Spark, transformation was a lazy operator; instead, a direct acyclic graph (DAG) of the RDD transformations was built, optimized, and only executed when an action was applied (Karau et al., 2015). Spark relied on a master to drive the execution of DAG on a set of executors (running on the Hadoop DataNodes). On top of the Spark transformation engine there was a suite of tool space: Spark SQL and data frames, Machine Learning Library (MLLib), Graph library (GraphX), and many third-party packages. These tools could be run interactively or via a batch job. Similar to Hadoop, all of

this software is available through the module environments package (Chrimes et al., 2016; Dhyani & Barthwal, 2014). Since the investigation was basically of healthcare user interactivity/usability, the following was elaborated on:

i) Spark terminal: Users can interact with Spark via its terminal. Similar to the usual command line terminals, everything was displayed as text. For Scala language (Java and Python), the user can call spark-shell to get access to the Scala (i.e., Python with Java libraries) terminal. For Python, the user should run Pyspark (module). For this studies platform, the scale mode of use was only recommended for advanced users. For all stakeholders, selecting a terminal is not user-friendly and the platform will be viewed as too cumbersome to use; therefore, it basically ran the module in simulation as an administrator with the end user.

ii) Jupyter: Jupyter is a successful interactive and development tool for data science and scientific computing. It accommodates over 50 programming languages via its notebook web application, and comprises all necessary infrastructures to create and share documents, as well as collaborate, report, and publish. The notebook was user-friendly and a rich document able to contain/embed code, equations, text (especially markup text), and visualizations. For utilization, the simultaneous running of all queries over the fast ingestion of the entire database was a significant achievement. Furthermore, corroboration of the results, after selecting a port and local host to run over a browser, started from 50 million and extending to three billion records, and validated the queries over the time of application to display. The existing software installations of Jupyter 4.0.6 on python 2.7.9 were implemented without customization.

iii) Zeppelin: Zeppelin was another interface used to interact with Spark. Zeppelin is a web-based notebook; it is not limited to Spark. It supports a large number of back-end processes through its interpreter mechanism (Karau et al., 2015). In Zeppelin, any other back-end process can be supported once the proper interpreter is implemented for it. SQL queries were supported via the SparkSQL interpreter and executed by preceding the query with a *%sql* tag. What makes Zeppelin interesting, from the HBDA platform perspective, was the fact that it has built-in visualizations and the user can, after running an SQL query, click on the icons to generate the graph or chart. This kind of at-the-fingertip visualization was essential for the zero-day or rapid analytics. Spark 1.5.2 and Zeppelin 0.6.0 were built from source, configured with a port and local host had to run over a browser, and used for the testing and benchmarking of the platform.

C. Drill

Inspired by Google's big query engine Dremel, Drill offers a distributed execution environment for large-scale ANSI-SQL:2003 queries. It supports a wide range of data sources, including *.csv*, JSON, HBase, etc... (Sitto & Presser, 2015). By (re)compiling and optimizing each of the queries while it interacting with the distributed data sets via the so-called Drillbit service, Drill showed capacity of the query with performance at a low latency SQL query. Unlike the master/slave architecture of Spark, in which a driver handles the execution of the DAG on a given set of executors, the *Drillbits* were loosely coupled and each could accept a query from the client (Dunning et al., 2016). The receiving Drillbit becomes the driver for the query, parsing, and optimization over a generated efficient, distributed, and multiphase execution plan; it also gathers the results back when the scheduled execution is done (Dunning et al., 2016; Jurney 2013). To run Drill over a distributed mode, the user will need a ZooKeeper cluster continuously running. Drill 1.3.0 and ZooKeeper 3.4.6 were installed and configured on the framework of the platform over a port with a local host.

4. Study Contributions/Results

4.1 Interview Results and Knowledge Transfer

Interviews were conducted and recorded at VIHA with business intelligence (BI) and data warehouse, clinical reporting, application platform services, database administrator, and health informatics architecture teams. During these interviews, patient encounter data from the hospital admission and discharge system were verified. Data profiles, dependencies, and the importance of the metadata for reporting performance were also emulated and verified. Current reporting limitations were recorded if the combinations of the DAD and ADT were done in one distributed platform running parallel queries. A total of 90 columns were confirmed as important to construct necessary queries and to combine ADT data with DAD data in the Big Data platform.

During the interviews and confirmation of the required columns to generate the emulation of aggregated data, all data elements and their required fields, as well as primary and dependent keys, were recorded and applied to the data generator in Excel and the Oracle 11g Express database (Table 7). All necessary data fields were populated for one million records before the replication of the emulated patient data to form up to three billion records.

The basic DAD elements derived from interviews before designing the SQL-like schema with Apache Phoenix (Figures 9 and 10):

Encounter ID (5 numeric digits)
PHN or Personal Health Number (10 numeric digits)
Patient Name (up to 25 characters)
Address (up to 50 characters)
Chief Complaint (text field)
Primary Physician Name (up to 25 characters)
Diagnosis (up to 25 characters)
Diagnosis Classification (ICD)
LOS or Length of Stay (numeric date)

Admission Date (month, day, year)
Discharge Date (month, day, year)
Discharge Disposition (up to 25 characters)

VIHA ADT core metadata housed by the BI Data Warehouse team is as follows:
a. Patient:
MRN or Medical Record Number - Health Authority number for each encounter
PHN – British Columbia Medical Service Provider
Name – Last, First, Middle
Date of Birth
Date of Death (time)
Address
Family Physician

b. Encounter:
Admin Type – Urgency or reason for visit
Admin Sources –patient location or not origin or birthplace
Encounter (numeric)
Admit (time) or Registration Date
Discharge Date/Time
Encounter Type – Inpatient, Outpatient, Surgical Daycare, Inpatient Extended
Medical Service (history and location) – Surgery, Orthopaedics, Laboratory, Medical Imaging (Radiology)
Discharge Disposition
Most Responsible Provider

c. Transfer:
Location – Fertility Clinic, Unit, Room, Bed Location
Transfer Date\Time

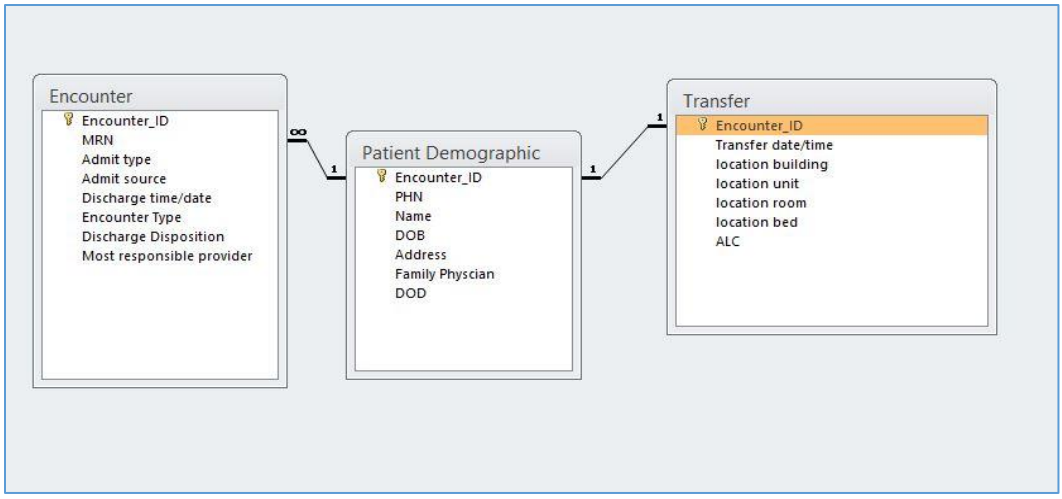

Figure 9. This study's part of the relational database of the admission, discharge, transfers (ADT) system at Vancouver Island Health Authority (VIHA).

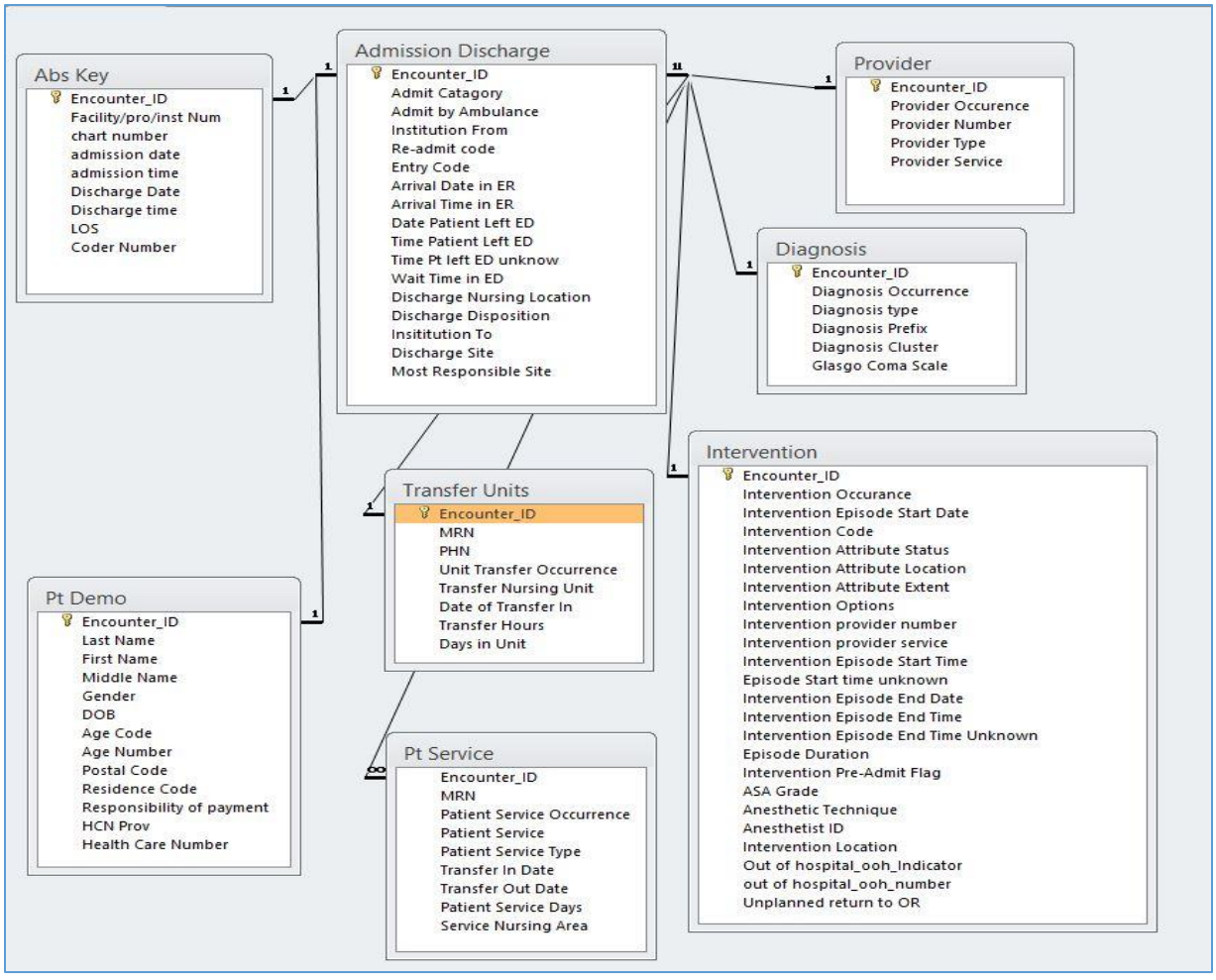

Figure 10. This study's part of the relational database of the discharge abstract database (DAD) system at Vancouver Island Health Authority (VIHA).

After final consultation with VIHA experts and deriving the randomized data, 90 columns were established (Table 8). This established combination of data elements and profiles of the metadata of ADT and DAD databases was then placed into SQL schema to run with Hadoop-MapReduce, as iterative bulkloads to form NoSQL database of HBase indexing (Table 9). Additionally, the queries derived were compared with clinical cases and how that interacted with the performance of the platform were representative of the clinical reporting at VIHA.

Table 8. Columns of the emulated ADT-DAD determined by interviews with VIHA.

DAD

Column Name	Data Type	
Reporting Prov	Number (5)	
MRN (Medical Record Number)	Varchar2(11)	a
Admission Date	Date(YYYYMMDD)	b
Admission Time	Time(hh:mm)	b
Discharge Date	Date(YYYYMMDD)	b
Discharge Time	Time(hh:mm)	b
LOS	Number (4)	b
Last Name	Text (10)	
First Name	Text (10)	
Middle Name	Text (10)	
Gender	Char(1)	
Birth Date	Date(YYYYMMDD)	b
Age	Number (3)	b
Postal Code	Char(7)	
Residence Code	Number(5)	
Responsibility for payment	Number(2)	
HCN Prov	Varchar2(2)	
Health Care Number	Varchar2(12)	a
Admit Category	Char(1)	
Admit by Ambulance	Char(1)	
Institute From	Number (5)	
Re-admit Code	Number (1)	
Entry Code	Char(1)	
Arrival Date in ED	Date(YYYYMMDD)	b
Arrival Time in ED	Time(hh:mm)	b
Date Patient Left ED	Date(YYYYMMDD)	b
Time Patient Left ED	Time(hh:mm)	b
Time Pt left ED unknown	Char(1)	b

a	Related to Primary Key
b	Related to Time/Date Calculations

Wait Time in ED	Time(hh:mm)	b
Discharge Disposition	Varchar2(2)	
Institution To	Number(5)	
Discharge Site	Char(1)	
Most Responsible Site	Char(1)	
Diagnosis Occurrence	Number(1)	
Diagnosis Type	Varchar2(1)	
Diagnosis Prefix	Varchar2(1)	
Diagnosis Code	Varchar2(7)	
Diagnosis Cluster	Char(1)	
Glasgow Coma Scale	Varchar2(2)	
Intervention Occurrence	Number(1)	
Intervention Episode Start Date	Date(YYYYMMDD)	b
Intervention Code	Varchar2(12)	
Intervention Attribute Status	Varchar2(1)	
Intervention Attribute Location	Varchar2(1)	
Intervention Attribute Extent	Varchar2(1)	
Intervention Options	Varchar2(1)	
Intervention Provider Number	Number(5)	
Intervention provider service	Number(5)	
Intervention Start Time unknown	Char(1)	b
Episode Duration	Number(4)	b
Intervention Pre-admit Flag	Varchar(1)	
Anesthetic Grade	Varchar2(01)	
Anesthetic Technique	Number(1)	
Anesthetist ID	Varchar2(5)	
Intervention Location	Varchar2(2)	
Out of Hospital_ooh_indicator	Char(1)	
Out of hospital_ooh_number	Number(5)	
Unplanned return to OR	Char(1)	
Provider Occurrence	Number(1)	
Provider Number	Number(5)	
Provider Type	Char(1)	
Provider Service	Varchar2(5)	
PHN	number(9)	a
Patient Service Occurrence	Number(1)	
Patient Service	Number (3)	
Patient Service Type	Char(1)	
Transfer In Date	Date(YYYYMMDD)	b
Transfer Out Date	Date(YYYYMMDD)	b

Patient Service Days	Varchar(5)	
Service Nursing Area	Varchar2(10)	
Unit Transfer Occurrence	Number(1)	
Transfer Nursing Unit	Varchar2(10)	
Date of Transfer In	Date(YYYYMMDD)	
Transfer Hours	Time(hh:mm)	
Days in Unit	varchar2(5)	b

ADT

Column Name	Data Type	
MRN (Medical Record Number	Varchar2(11)	a
FIN (Facility ID Number)	Varchar2(8)	a
Encounter Number	Varchar2(10)	a
Admit Type	Varchar2(1)	
Admit Source	Varchar2(10)	
Admit Time	Time(hh:mm)	b
Admit Date	Date(YYYYMMDD)	b
Discharge time	Time(hh:mm)	b
Discharge date	Date(YYYYMMDD)	b
Encounter Type	Varchar2(10)	
Medical services	Varchar2(25)	
Discharge Disposition	Varchar2(2)	
Most Responsible Provider	Varchar2(25)	
PHN	Number(9)	a
Last Name	Text (10)	
First Name	Text (10)	
Middle Name	Text (10)	
DOB	Date(YYYYMMDD)	
Address	Varchar2(25)	
Family Physician	Varchar2(25)	
DOD	Date(YYYYMMDD)	b
DOD time	Time(hh:mm)	b
Transfer date	Date(YYYYMMDD)	b
Transfer TIME	Time(hh:mm)	b
Location building	Varchar2(4)	
Location unit	Varchar2(4)	
Location room	Varchar2(4)	
location bed	Varchar2(1)	

a	Related to Primary Key
b	Related to Time/Date Calculations

Table 9. Established data schema in HBase-Phoenix required for Bulkloading.

```
CREATE TABLE IF NOT EXISTS DADS1 (
EncounterID   BIGINT NOT NULL,
Admit_by_Ambulance VARCHAR,
Admit_Category VARCHAR,
Admission_Date VARCHAR,
Admission_Time VARCHAR,
Age INTEGER,
Anesthestic_Grade  VARCHAR ,
Anesthetist_ID  VARCHAR,
Anesthetistic_Technique  INTEGER,
Arrival_Date_in_ER  VARCHAR NOT NULL,
Arrival_Time_in_ER  VARCHAR NOT NULL,
Date_Patient_Left_ED  VARCHAR ,
Date_of_Transfer_In  VARCHAR,
Days_in_Unit  VARCHAR,
Discharge_Date VARCHAR NOT NULL,
Discharge_Disposition VARCHAR NOT NULL,
Discharge_Site VARCHAR NOT NULL,
Discharge_Time VARCHAR NOT NULL,
Birth_Date  VARCHAR,
Diagnosis_Cluster VARCHAR,
Diagnosis_Code VARCHAR NOT NULL,
Diagnosis_Occurrence  INTEGER,
Diagnosis_Prefix VARCHAR,
Diagnosis_Type VARCHAR,
Entry_Code VARCHAR,
Episode_Duration  INTEGER,
First_Name VARCHAR,
Glasgo_Coma_Scale VARCHAR,
Gender  VARCHAR,
Health_Care_Number VARCHAR NOT NULL,
HCN_Prov VARCHAR,
Institute_From  INTEGER,
Institution_To  INTEGER,
Interven_Attribute_Extent VARCHAR,
Interven_Attribute_Location VARCHAR,
Interven_Attribute_Status VARCHAR,
Interven_Code VARCHAR,
Interven_Episode_Start_Date  VARCHAR,
Interven_Episode_St_Date  VARCHAR,
Interven_Location  VARCHAR,
Interven_Occurrence  INTEGER,
Interven_Options VARCHAR,
```

Interven_Pr_Number INTEGER,
Interven_Preadmit_Flag VARCHAR,
Interven_provider_service INTEGER,
Interven_Start_Time_unknown VARCHAR,
Interven_St_T_unknown VARCHAR,
Last_Name VARCHAR,
LOS INTEGER NOT NULL,
Middle_Name VARCHAR,
Most_Responsible_Site VARCHAR,
MRN VARCHAR,
Out_of_Hospital_ooh_indicator VARCHAR,
Out_of_hospital_ooh_number INTEGER,
PHN INTEGER,
Postal_Code VARCHAR,
Pdr_Number INTEGER,
Provider_Occurrence INTEGER,
Provider_Service VARCHAR,
Provider_Type VARCHAR,
Patient_Service INTEGER,
Patient_Service_Days INTEGER,
Patient_Service_Occurrence INTEGER,
Patient_Service_Type VARCHAR,
Readmit_Code INTEGER,
Reporting_Prov INTEGER,
Residence_Code INTEGER,
Responsibility_for_payment INTEGER,
Service_Nursing_Area VARCHAR,
Time_Patient_Left_ED VARCHAR,
Time_Pt_left_ED_unknown VARCHAR,
Transfer_Hours VARCHAR,
Transfer_Nursing_Unit VARCHAR,
Transfer_In_Date VARCHAR,
Transfer_Out_Date VARCHAR,
Unit_Transfer_Occurrence INTEGER,
Unplanned_return_to_OR VARCHAR,
Wait_Time_in_ED VARCHAR,
FIN INTEGER NOT NULL,
Encounter_Number INTEGER NOT NULL,
Admit_Source VARCHAR,
Encounter_Type VARCHAR,
Medical_Services VARCHAR,
MostResponProvider VARCHAR,
Address VARCHAR,
Family_Physician VARCHAR,
Location_Building VARCHAR NOT NULL,
Location_unit VARCHAR NOT NULL,

Location_Room VARCHAR NOT NULL,
Location_Bed VARCHAR NOT NULL,

4.2 Technical Implementation

i. General Steps

General steps that were started when the head node at WestGrid was accessed to start PBS job.

A. Here are the first steps of the job initialization to run the different modules to ingestion of HFiles to HBase bulkloads were as follows:

1- qsub -I -l walltime=72:00:00,nodes=6:ppn=12,mem=132gb
2- ll /global/software/Hadoop-cluster/ -ltr
(use the latest in that folder at the bottom – hdp 2.6.2, hb 0.98.16.1, phoenix 4.6.0)
3- module load Hadoop/2.6.2
4- setup_start-Hadoop.sh f (f for format; do this only once…).
5- Module load HBase/…
6- Module load phoenix/…
7- (actually check the ingest.sh script under ~/bel_DAD)
8- hdfs dfsadmin -report
9- djps (command displays the JVMs, Java services running with PIDs)

B. To use the existing scripts, then one of them was d_ingest.sh to use as template.

Here was the process to ingest the file into a phoenix/HBase database:
1- module load Hadoop/2.6.2
2- module load HBase/0.98.16.hdp262
3- module load phoenix/4.6.0
4- localFileName=”The CSV file containing your data”
5- hdfs dfs -mkdir /data
6- hdfs dfs -put "$localFileName" /data/
7- hdfs dfs -ls /data
8- sqlline.py hermes0090-ib0 DAD.sql
9- export HADOOP_CLASSPATH=/global/software/Hadoop-cluster/HBase-0.98.16.1/lib/HBase-protocol-0.98.16.1.jar:/global/software/Hadoop-cluster/HBase-0.98.16.1/lib/high-scale-lib-1.1.1.jar:/global/scratch/dchrimes/HBase-0.98.16.1/34434213.moab01.westgrid.uvic.ca/conf
10- time Hadoop jar /global/software/Hadoop-cluster/phoenix-4.6.0/phoenix-4.6.0-HBase-0.98-client.jar org.apache.phoenix.MapReduce.CsvBulkLoadTool --table DAD --input "/data/$localFileName"
#psql.py -t DAD localhost all.csv

C. Ingest all using d_runAll.sh

1- First decide which file to use, then check the correctness of its column names. DADV2.sql (for v2) and DAD.sql (for old)

2- Create the database table using sqlline.py as illustrated above (sqlline.py hermes0090-ib0 DAD.sql)
3- Make sure all the modules loaded:
module load Hadoop/2.6.2
module load HBase/0.98.16.hdp262
module load phoenix/4.6.0
4- Generate the rest of data (we need 10 billion/ 4billions).
5- Use the d_runAll.sh to ingest them all at once.
6- If a problems happen (persist) check the logs in different location (/global/scratch/dchrimes/ and/or on the /scratch/JOBID on the nodes).

ii. Hadoop-HBase-Phoenix –Technical Implementation and Challenges

After installing Hadoop's ecosystem from source on the cluster (with dedicated and hard local disks at 40TB over the 6 nodes), a dynamic Hadoop job (HDFS version 2.6.0) was launched that functioned effectively over the 6 nodes at WestGrid. The platform worked as expected after modified configurations of Hadoop's *hdfs-site.xml* and part of the MapReduce frame, as well as additional configurations of *mapred-site.xml*. This ran under the Hadoop resource manager, Yarn (with configuration file *yarn-site.xml*), which also required additional configurations in the xml to not be defaulted to network connections and to not timeout. Additionally, the number of replicas was set to three in the xml with connection to InfiniBand or *ib0*. To interact with HDFS, command scripts were run to automate the ingestion step (generate data replication in the exact format specified by SQL script to the nodes). In the xml script, there were slight modifications after several iterations of 50 million (258MB) of simulated tested and to have the time optimized at 200-300 minutes for each iteration. Furthermore, there was a lot of investigation and blog chats to achieve this and initially it was tried for different size of file of 100 million and more but the fastest time was achieved at 259 minutes with compression of the HBase files run afterwards between the Hadoop-Reduce ingestions. Compression reduced the performance by ~100 minutes from 399 to 299 minutes on average.

The Map part of the platform showed high performance at 3–10 minutes, but the Reduce took 3–12 hours; however, a Reducer time of 3–4 hours was achieved once configurations in Yarn, ZooKeeper, and others were either enabled or disabled and worked over the platform as expected, relying on InfiniBand bandwidth at low latency over WestGrid.

HBase (NoSQL version 0.98.11) was composed of the main deployment master (DM) and fail-over master, the RegionServers holding HBase data, and a ZooKeeper of five nodes to orchestrate the ensemble, called *RegionServers*. It was tested with three and five worker nodes, the latter exhibited higher performance. The xml configuration file *HBase-site.xml* and *HBase-env.sh* were adjusted to tune the performance of HBase. In fact, it was found that the queries of Apache Phoenix or Apache Spark resided as a layer on HBase; therefore, there were additional adjustments for Spark after the performance stabilized for Phoenix. Moreover, there was also *short-circuiting* xml configuration enabled.

Apache Phoenix (version 4.3.0), a thin layer on top of HBase was used as ingest structured file and schema-based data into the NoSQL database. It can run SQL-like queries against the HBase data and output example provide in Table 10.

It is important to note that queries included a range Encounter IDs, in this below example, i.e., EncounterID<1000, because it was discovered early on that the replicated data showed inconsistent query efficiency (performance) times. Without the EncounterID (from HBase indexing) the queries and the family columns could not be fully tested, and, the results would be confounded as the replication increased such that the performance was not related to the platform but the replication. It was important to compare the same range over the larger volume to benchmark performance. Otherwise, the larger the volumes with greater replications would exponentially increase the time duration to receive the query result.

Table 10. The following is an example from SQL-Phoenix queries with commands and outputs via python GUI on WestGrid's interface.

```
sqlline.py hermes0090-ib0
0: jdbc:phoenix:hermes0090-ib0> select * from DAD where EncounterID<1000 and
ADMIT_BY_AMBULANCE='N';
+-----------------------------------------+-----------------------------------------+-----------------------------
-------------+-----------------------------------------+-----------------------------------------+-------------
------+
|            ENCOUNTERID                  |           ADMIT_BY_AMBULANCE            |
ADMIT_CATEGORY              |             ADMISSION_DATE              |
ADMISSION_TIME              |          AGE          |
+-----------------------------------------+-----------------------------------------+-----------------------------
-------------+-----------------------------------------+-----------------------------------------+-------------
------+
| 3                         | N                         | L                         | 2000-06-18
| 03:32                       | 36              |
| 5                         | N                         | S                         | 1979-08-30
| 18:17                       | 45              |
| 6                         | N                         | R                         | 1948-05-13
| 16:15                       | 5               |
| 8                         | N                         | L                         | 2014-08-14
| 09:42                       | 9               |
| 14                        | N                         | R                         | 2006-06-23
| 14:42                       | 81              |
| 15                        | N                         | R                         | 1939-02-10
| 13:30                       | 99              |
| 16                        | N                         | S                         | 1941-03-21
| 07:10                       | 6               |
| 17                        | N                         | R                         | 1954-07-18
| 05:24                       | 2               |
| 19                        | N                         | U                         | 1948-12-16
| 08:02                       | 53              |
| 20                        | N                         | N                         | 2003-10-02
| 18:48                       | 43              |
| 21                        | N                         | L                         | 1981-05-03
| 17:19                       | 35              |
```

25	N	R	2002-05-09	02:54	77
26	N	L	1979-12-06	18:02	90
27	N	R	1949-11-22	02:29	45
29	N	R	1994-04-05	02:33	25
31	N	N	1980-06-30	0.0247	84
32	N	N	1949-01-19	10:28	73
34	N	S	1995-02-09	18:19	60
38	N	S	1989-06-13	08:33	69
40	N	S	1969-05-21	15:41	10
41	N	R	1961-04-10	04:52	43
44	N	L	1939-06-02	09:38	11
46	N	L	1970-06-18	05:31	81
47	N	N	1979-08-14	10:48	83
48	N	R	1961-07-10	19:38	9
50	N	S	2007-03-15	08:26	36
51	N	N	1933-05-17	11:36	15
53	N	U	1989-03-24	22:33	59
54	N	R	1930-01-20	15:16	46

470 rows selected (1.891 seconds)

0: jdbc:phoenix:hermes0090-ib0> select * from DAD where EncounterID<50000 and ADMIT_BY_AMBULANCE='N' and ADMISSION_DATE='1986-07-31';

```
| ENCOUNTERID | ADMIT_BY_AMBULANCE | ADMIT_CATEGORY | ADMISSION_DATE | ADMISSION_TIME | AGE |
+-------------------------------------+-------------------------------------+-------------------------------------+-------------------------------------+-------------------------------------+-------------------------------------+
| 9948 | N | L | 1986-07-31 | 05:46 | 74 |
+-------------------------------------+-------------------------------------+-------------------------------------+-------------------------------------+-------------------------------------+-------------------------------------+
1 row selected (1.211 seconds)
0: jdbc:phoenix:hermes0090-ib0>
```

It was found that Phoenix is similar to the HBase shell and equipped with a python interface to run SQL statements. A *.csv* file was placed into the bulkloader to ingest the large flat file using MapReduce. The load balancing between the RegionServers (i.e., "salt bucket") was set to the number of slaves necessary to balance the ingested data evenly. The constraint run in the ingestion that represented the primary keys to emulate dependencies and balanced ingestion of the *.csv* file database was:

CONSTRAINT PK PRIMARY KEY (EncounterID, Arrival_Date_in_ER, Arrival_Time_in_ER, Discharge_Date, Discharge_Disposition, Discharge_Site, Discharge_Time, Diagnosis_Code, Health_Care_Number, OS, FIN, Encounter_Number, Location_unit), Salt Buckets = 5

To improve the ingestion of the one billion rows and 90 columns to attempt to generate 1-10 billion rows, local disks of 40TB in total were physically installed on the worker nodes. After local disks were installed on five (worker) nodes, a set of shell scripts was used to automate the generation and ingestion of 50 million records at each of the iterations via MapReduce. The goal of study was to run 10 billion rows but the maximum achieved was 3 billion due to operational barriers, workflow limitations and table space because key stores almost tripled the amount of space used for each of the ingestions (Table 11). Some limitations of the Hadoop/MapReduce framework partly occurred because local disks had to be reinstalled after the failover from WestGrid's 500GB disks to 2TB slots failed. The entire data that was achieved at 3 billion had to be rerun. In total, including all the testing that did achieve about 6-9 Billion rows were ingested to the local disks in iteration of which three billion were correctly indexed and could be queried.

Other findings of big data technology limitations installed on WestGrid's architecture was ongoing manually intervention was required to constantly fine tune the performance of bulkloads from MapReduce to HBase. Hadoop had ingestions exhibiting high performance, circa three minutes to complete task for 258MB or each 50 million rows. Sometimes HDFS was unbalanced and had to be re-run to re-balance the data to the nodes or when the local disk at 500GB did not failover to 2TB disks installed, the entire ingestions had to start all over again because HBase could not re-index and, therefore, its queries were invalid with no indexes, which drastically slowed performance when not operational (Table 11). In fact, with the continued re-running the

ingestions over the course of two years, 9-10 billion rows were achieved; however, only three billion were established with the correct indexed rows for each unique row.

The task scheduler for MapReduce had the map portion taking only 3 minutes, but the Reduce part of MapReduce bulkload took 3–12 hours. Map started much earlier than Reduce, and while there were some background services running before Reduce ran (to start it faster and to run parallel with Map), Reduce did not run at a constant percentage of Map and ranged from 20–60% (Figure 11). Reducer was taking a very long time to run bulkloads to HBase and this was due to the SQL schema and a known bottleneck in the combination of MapReduce and HBase programming. Blogs were sent with the issue and detail to ask how to break/fix or if this a known issue over several online programming sites of Apache Hadoop and HBase. Others confirmed that ingesting into HBase via MapReduce took longer than expected and no resolution could be found with the new and old versions of HBase. This issue was due in part to RegionServer hot-spotting or over-utilization and stalled releasing of the process, even with the use of a salted table. It was also not that longer periods of times occurred when the reducer did not start at ~20% of the Map so processes where not running in parallel between Map and Reduce. The platform had a major limitation with the MapReduce that persisted more when ingesting files to HBase (Table 11). ZooKeeper and Yarn connectivity and RegionServers connecting to network only persistently connected to UVic's network instead of WestGrid's InfiniBand, which drastically slowed performance. Some configuration to ZooKeeper and Yarn reduced this issue but no resolution with HBase other than manually running compaction in iteration, which required an additional 3-4 hours at each of the iterations) was found. When compaction of HBase Files was done, it was noticed that Reducer more consistently started at 20%. It did not start at the same time as Map and this was due to the fast Map job that was completed quickly before each percent 0 to 1 to 2 to 3 etc… was executed and incrementally completed in its scheduler.

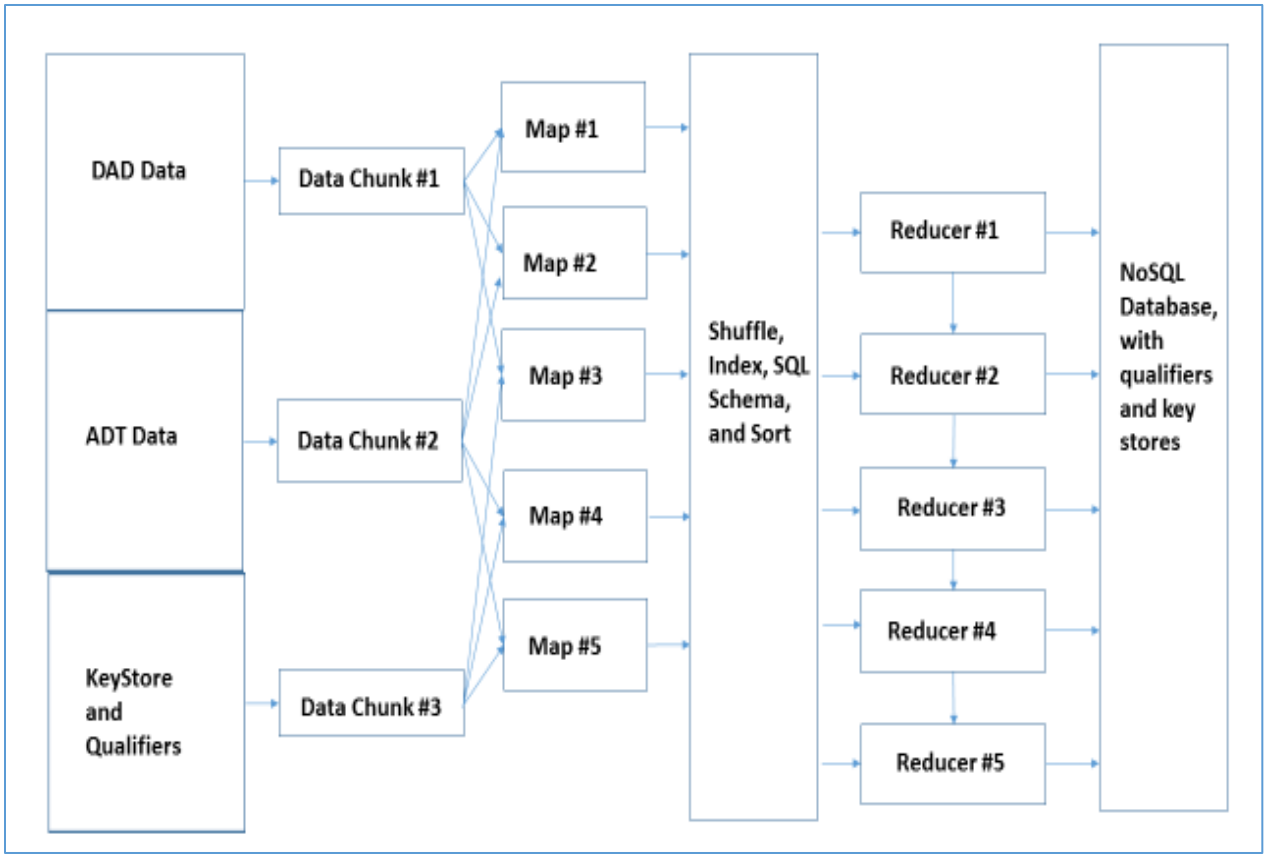

Figure 11. Possible MapReduce transaction over the proposed BDA platform.

iii. Hadoop-HBase-Phoenix –Efficiencies and Outputs

Mainly the long durations of ingesting data were found to be relation to the HBase's RegionServers. This delay was due to the HBase with the Reducer, *configureIncrementalLoad*, which ensures a total ordering and the necessary sorting of the key-values to produce the proper HFiles. The loading of HFiles into HBase was quick and it took only a few seconds. It was attempted to run ingestions with non-salted tables and that took more than three days for a single iteration to run and justified the incorporated use of the salted tables in SQL code via Phoenix to load into HBase schema and nodes. The length of the column names in the database scheme to bulkloads in HBase reduced performance and this was optimized (Table 11).

There were some findings on optimized performance of the platform. For instance, the physical memory consumption (in pink) and the cached memory (in blue) depicted in Figure 12 shows that an ideal high performance at one iteration to generate 50 million. This was compared to Figure 13 compared to iteration run with extremely long ingestions or too short ingestions due to dead RegionServers and extremely unbalanced ingestions with some nodes reaching 90% while others only 20% full table space, at the same iteration at 50 million over several times. Moreover, the increased activity in mid-May to October 2016 to complete the three billion shows the 24GB RAM constantly pinged at its peak performance, which was what the technical specifications said is a good indicator of high performance and stable platform (Figure 14). There were also two additional indicators of the performance of the platform while the ingestions

are running. CPU usage needs to be maxed, which is during mid-May to October 2016 it pinged at 100% but did not stay due to running compaction after each of the ingestions and took over 4 hours (Figure 15). And, the IO disk usage needs to reach the best possible throughput provided, which showed 160MB/s was achieved and pinged at approximately the same time at the peak performance of the corresponding ingestions (Figure 16).

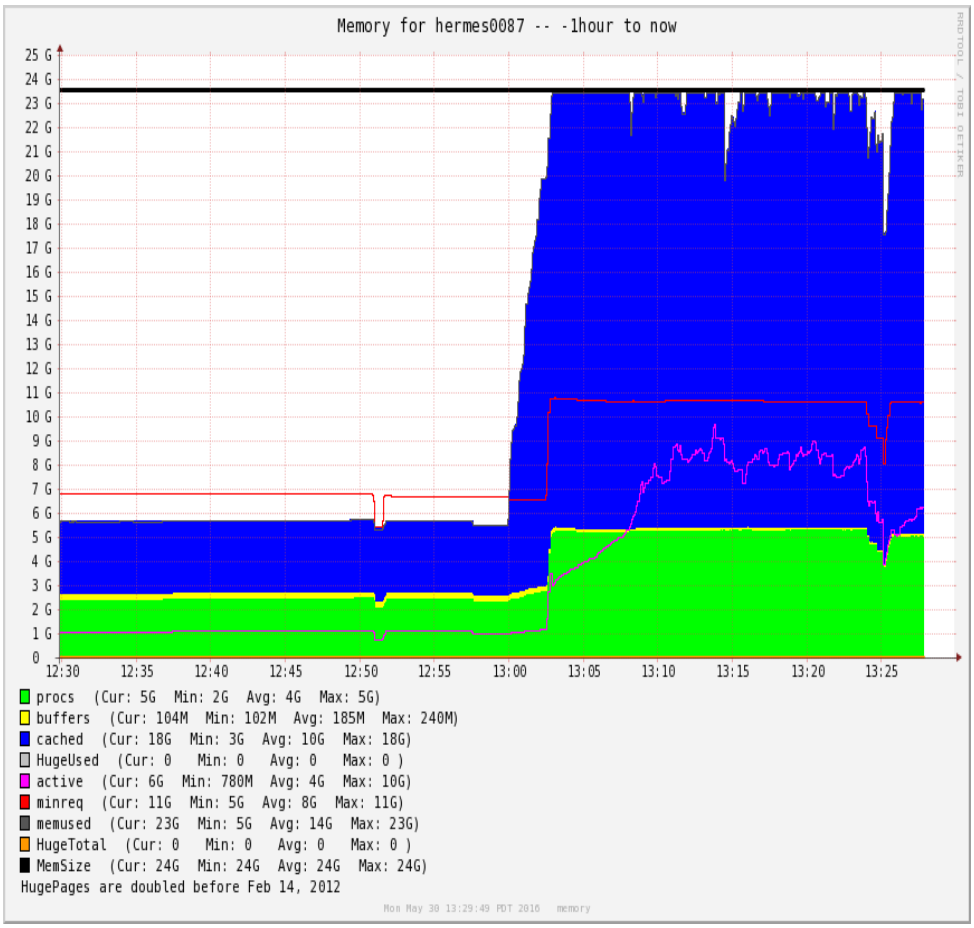

Figure 12. A high performance ingestion run with 24GB ram consumption on Hermes87 node reported from WestGrid showing variation in the duration of the ingestion of 50 Million records over each of the iterations over the two weeks. The graph shows the following cached memory (in blue), active passes (in green), buffers (in yellow), as well as active (in pink), minimum required (in red) and available memory (black line).

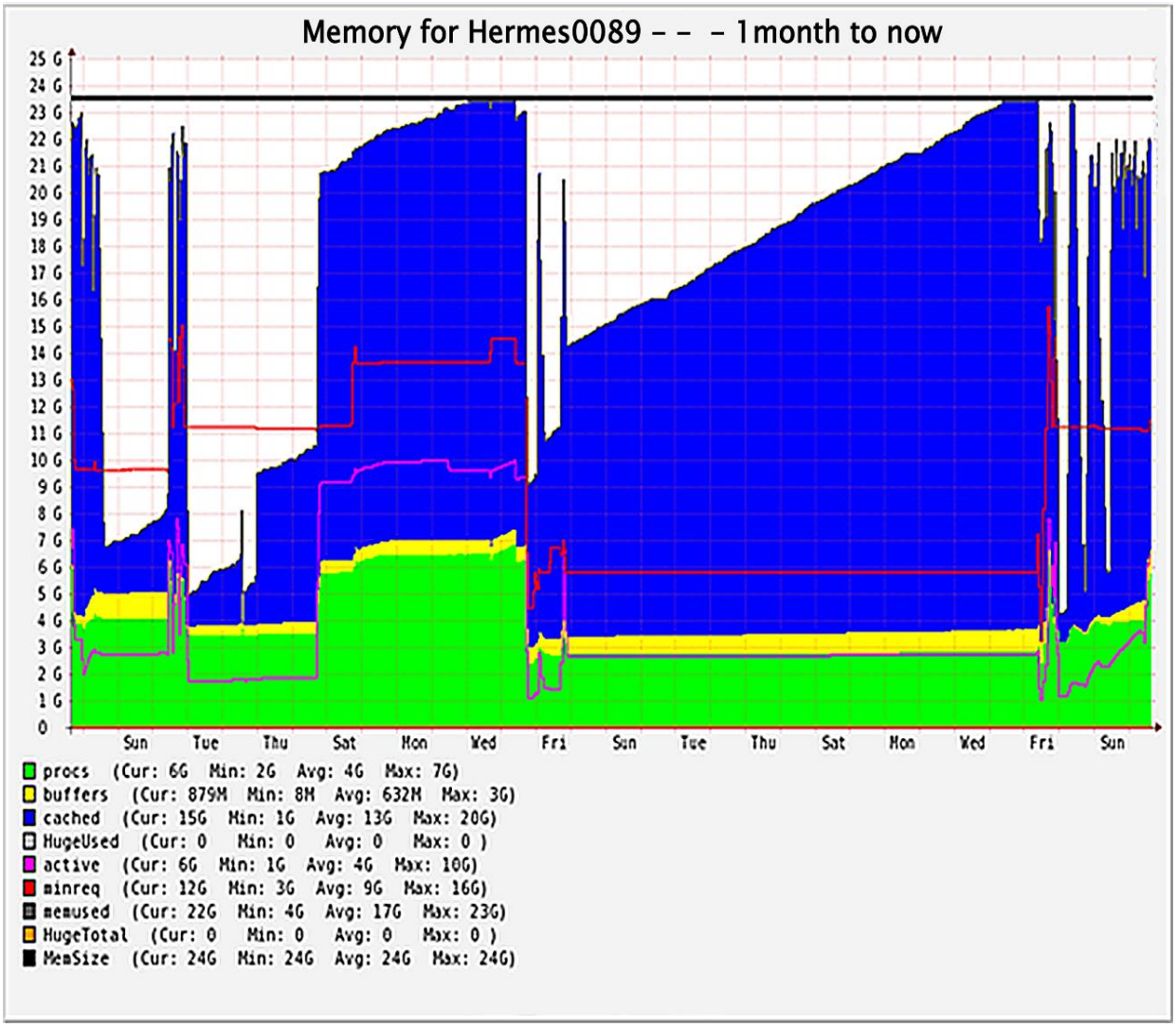

Figure 13. A month of varied iteration lengths with 24GB ram consumption on Hermes89 node reported from WestGrid showing variation in the duration of the ingestion of 50 Million records over each of the iterations over a month. The graph shows the following cached memory (in blue), active passes (in green), buffers (in yellow), as well as active (in pink), minimum required (in red) and available memory (black line).

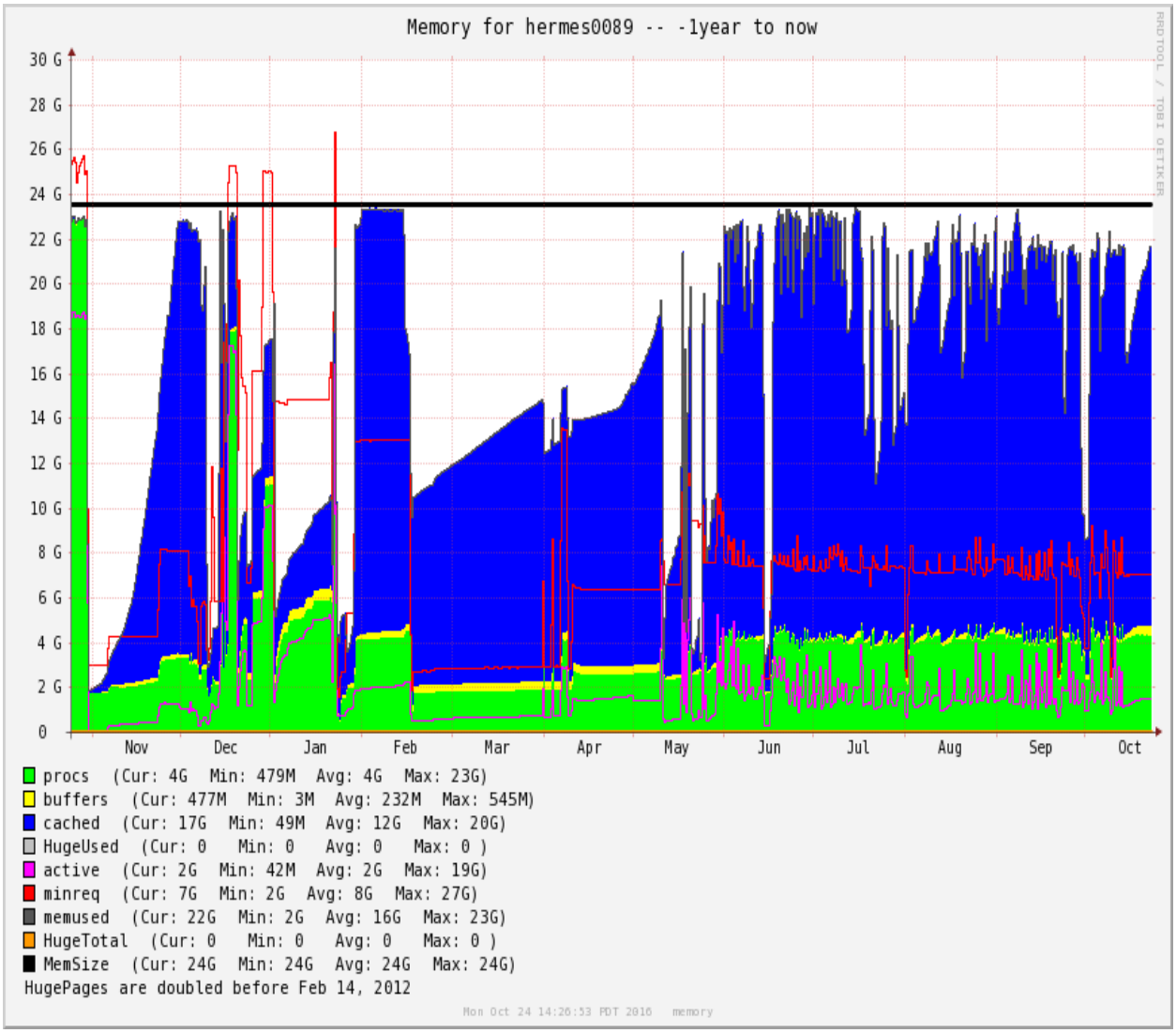

Figure 14. A month-to-month varied iteration lengths with 24GB ram consumption on Hermes89 node reported from WestGrid showing variation in the duration of the ingestion of 50 Million records over each of the iterations over an entire year (November 2015 to October 2016), with more activity mid-May to October. The graph shows the following cached memory (in blue), active passes (in green), buffers (in yellow), as well as active (in pink), minimum required (in red) and available memory (black line).

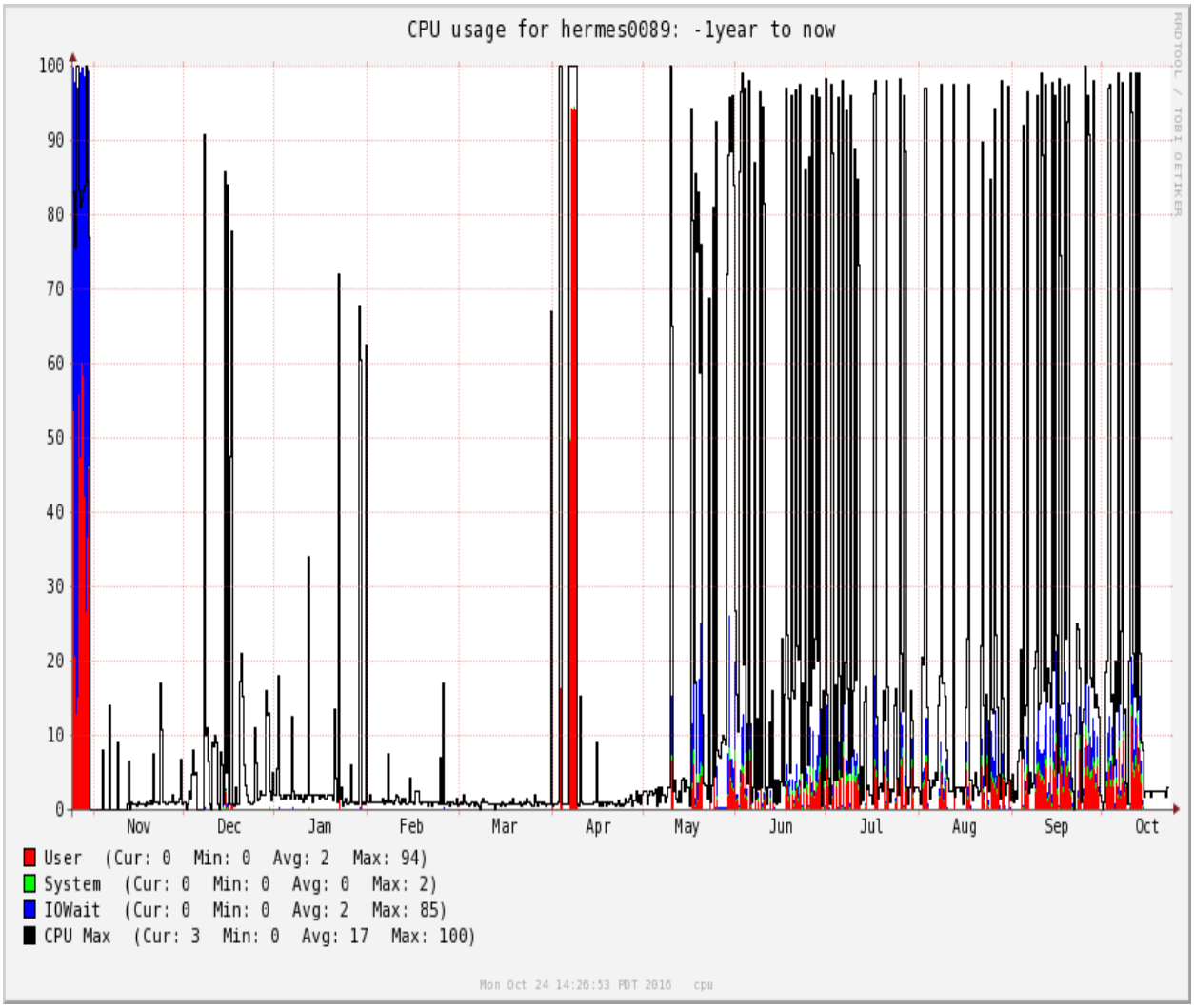

Figure 15. A year of varied iteration and CPU Usage (at 100%) on Hemes89 node reported from WestGrid showing variation in the duration of the ingestion of 50 Million records over each of the iterations. The graph shows the following, user (in red), system (in green), IO Wait time (in blue), and CPU Max (black line).

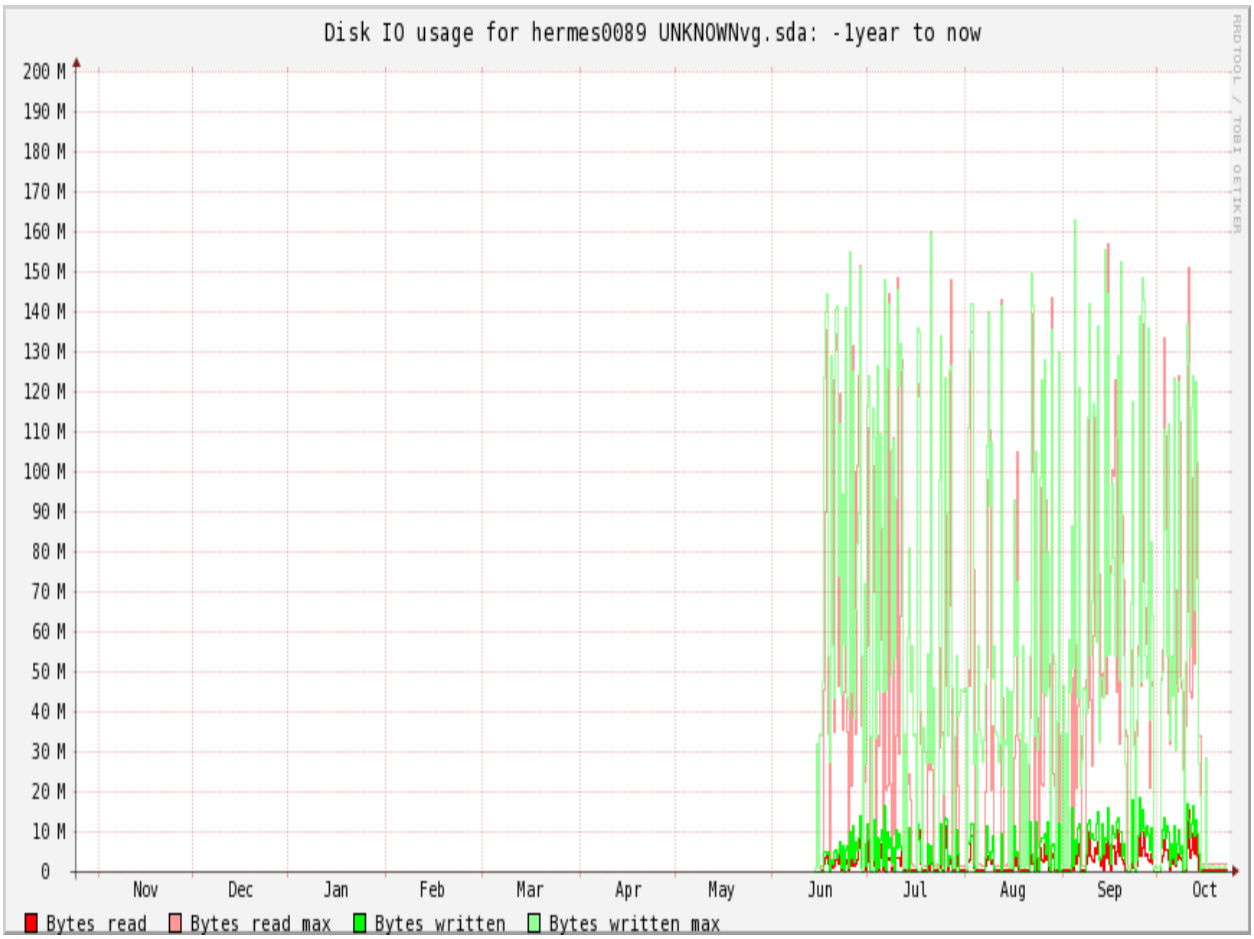

Figure 16. A year of varied iteration and IO Disk utilization (at up to 200MB/s) on Hemes89 node reported from WestGrid showing variation in the duration of the ingestion of 50 Million records over each iteration daily during mid-May to October, 2016. The graph shows the following, bytes read (in red), bytes read max (in light read), bytes written (in green), and bytes written max (in light green). The bytes written and its max represent the performance of the InfiniBand (160MB/s was achieved).

The main building block of the BDA platform was a functional dynamic Hadoop ecosystem comprised of Hadoop, HBase, Phoenix, Spark, Drill, Zeppelin, and Jupyter. These packages were built and customized to offer high performance, as well as adhere to the existing architecture of the WestGrid cluster. In order to corroborate with the previously reported results and experience with HBase/Hadoop and Phoenix (Chrimes et al., 2016; Kuo et al., 2015) dedicated this paper to Spark, Zeppelin, and Drill to investigate how Spark and Drill would directly interact with Hadoop over the platform while running the emulated data ingested into the Hadoop file system in its simplest form, commonly used *.csv* format or flat file (Chrimes et al., 2016). The results showed that Drill outperformed Spark/Zeppelin; however, it was restricted to running ANSI-SQL-like queries against the data. Zeppelin running on Spark showed ease-of-use interactions, though it lacked the flexibility offered by Jupyter on Spark (which was at the expense of requiring extra setup time and extra lines of contextual code to start the queries). Zeppelin and Jupyter on Spark offered high performance stacks not only over the platform but also to collaborate, share, and publish results with other stakeholders. Furthermore, the ingestion

time of one billion records was circa two hours via Apache Spark compared to >24 hours for HBase/Phoenix, indicating a performance bottleneck if HBase needs to run ingestions.

The big data platform in its construct was evaluated on the open source software available. HBase was chosen due to its NoSQL services, maintenance of indexing flat files during Hadoop ingestions, and allowing for SQL-like layers via Apache Phoenix to code in the required columns via specified salt buckets to ingest a csv flat file to form a NoSQL database. Apache Spark can also be constructed as a separate service on nodes in conjunction with HBase services as well as Hadoop's. This is known as the dynamic Hadoop module; it was constructed on HBase to replicate the data and also to query the data via a python interface using Phoenix.

The construct of master and worker nodes in the services by HBase and Hadoop was due to the fact that the original architecture of the supercomputing nodes at WestGrid has similar architecture; therefore, a deployment node in the build of the dynamic Hadoop cluster was required. WestGrid also incorporates an InfiniBand latency of connectivity of >40GB/s, which had to be configured on HBase and Hadoop's deployment manager, known as Yarn. The InfiniBand was used because it offers low-latency and high-bandwidth connectivity between the nodes. HBase was easy to configure with InfiniBand but Hadoop's deployment manager services and its nodes did not always utilize the InfiniBand. Some nodes at WestGrid had greater connectivity than others, which caused an imbalance of the files in how the table space was updated on each of the files. This caused a maximized space on the node with greatest connectivity. To solve this problem, it was added to the SQL as a salt bucket function that called in the sequence of SQL-like ingestion code of the bulkload from MapReduce to HBase, and it was done in conjunction with the short-circuit xml enabled. After that, by in large, the iteration populated the nodes with table space in balance. There is a tool called Hannibal available to check Regions over Hadoop/HBase and showed that the five Regions averaged ~1TB (Figure 17) and this coincided with 176-183 Store files for each (comprised of Hadoop's HFiles) for the entire distributed table (ingested into HBase) for three billion records (Figure 18). Total equals 900 Regions and 900 Store files. The slight imbalance and an initial peak in the sizes because compaction was disabled to run automatically (due to performance issues) and manually running bot minor (combines configurable number of smaller HFiles into one larger HFile) and major reads the Store files for a region and writes to a single Store file) compaction types. Furthermore, the durations of the iterations still continued to fluctuate with no distinctive trend for one billion (Figure 19). For three billion, at more operational level at high core CPU over May-October 2016, had similar fluctuations of 299-1600 minutes with average of ~300-375 minutes for ingestions and roughly the same amount of time for compaction (major and minor) manually run before the next ingestion iteration.

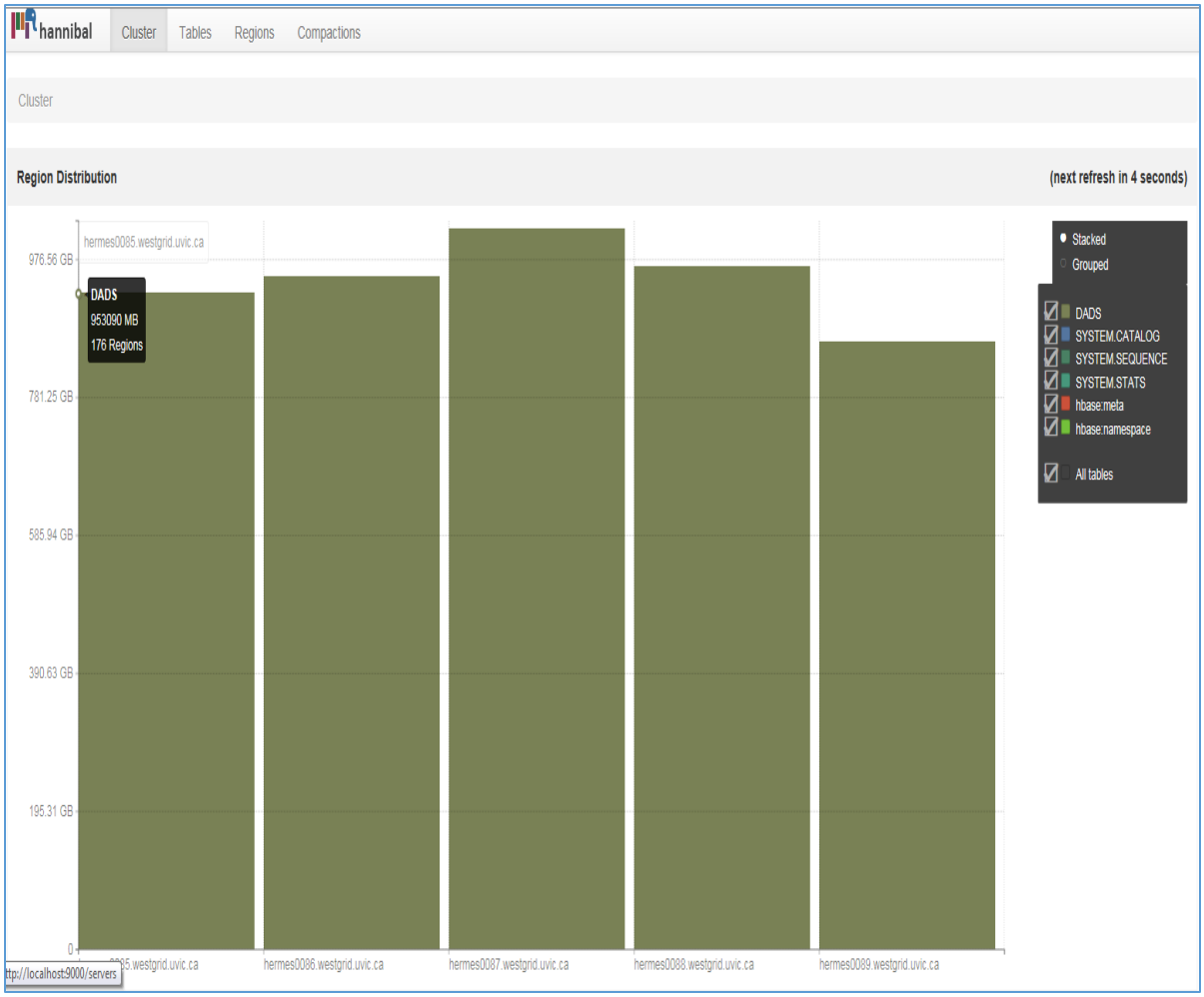

Figure 17. The five worker nodes with each 176-183 Regions in HBase and 1-5TB for the complete ingestion of three billion records. Hannibal is a tool inherent to Hadoop/Base that can produce statics on Regions in the database nodes.

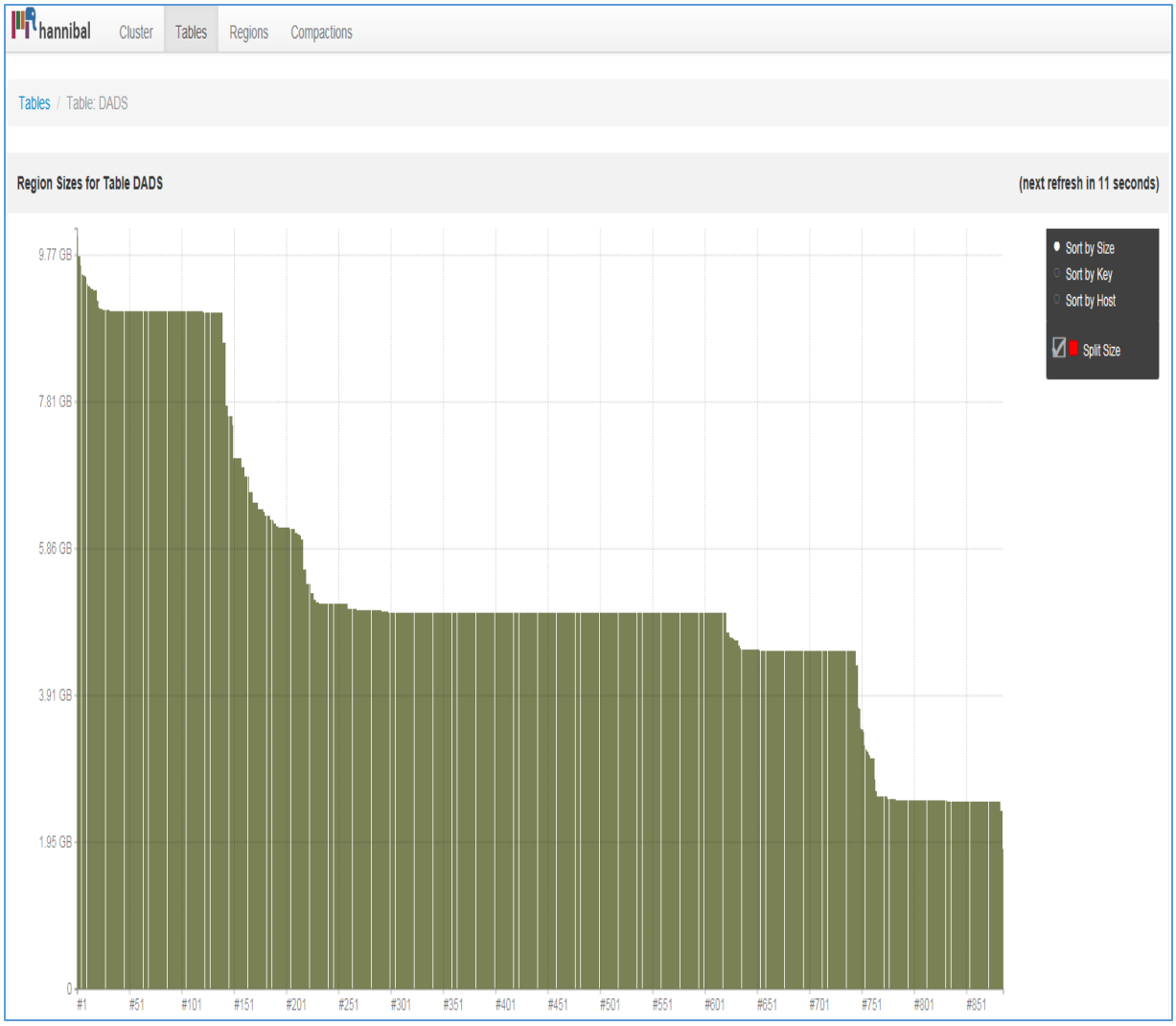

Figure 18. Region sizes of the entire table for the three billion records that totalled 851 Store files, which contain Hadoop's HFiles. There is an initial peak in the sizes because compaction was disabled to run automatically (due to performance issues) and manually running bot minor (combines configurable number of smaller HFiles into one larger HFile) and major reads the Store files for a region and writes to a single Store file) compaction types.

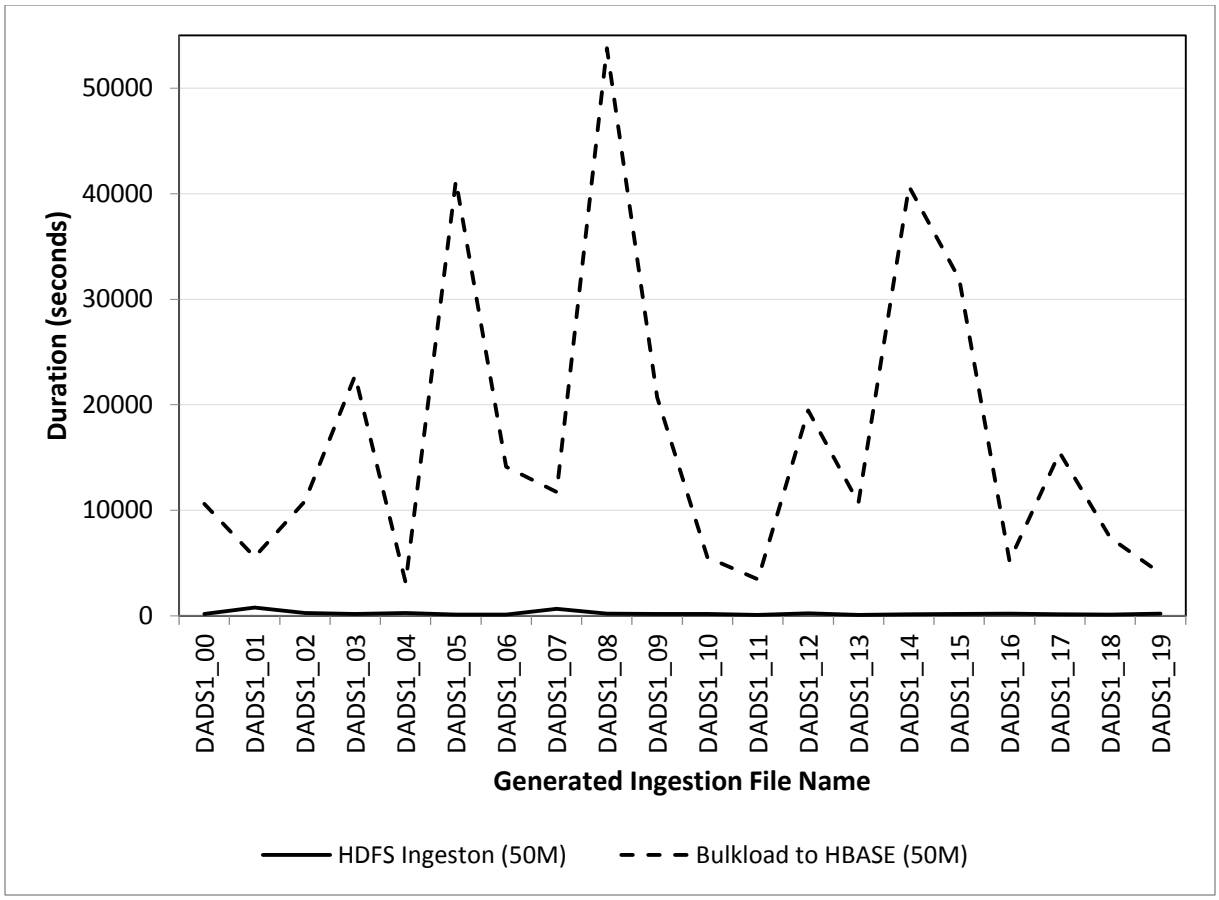

Figure 19. Duration (seconds) of the generated ingestion files at 50 each to reach 1 billion that include the Hadoop ingestion of HFiles (solid line) and HBase Bulkloads (dotted line).

iv. Hadoop-HBase-Phoenix –SQL Queries

The deployment of the Hadoop environment on the nodes was carried out behind the backend database scenes via a sequence of setup shell scripts that the user needs to call after loading the necessary Hadoop modules. These setup scripts created an initial configuration of bringing in the distributed files directly and were highly dependent on the number of nodes chosen when launching the job. The user can then adjust those configurations to match the needs of the job and its performance. Currently, with six nodes, one master node was deployed and the rest acted as slaves or workers. The services that the master node and slave nodes run and this made the consistent hitting of the enabled InfiniBand when ingesting a challenge. It was found that most of the Hadoop environment services rely on the hostname to get the IP address of the machine. Since the hostnames on a cluster were usually assigned to their management network, the setup scripts and the configuration files needed to be constantly adjusted. Yarn, Hadoop's resource and job manager, unfortunately still used the Gig-Ethernet interface when orchestrating between the nodes; however, the data transfer was confirmed to be carried out on the InfiniBand.

It is worth emphasizing that the platform had to rebuild Hadoop, Phoenix, and HBase from source. This build was necessary to enable the Hadoop native libraries (more efficient services to ingest files and query since Hadoop needs to continually run), which are especially used to

efficiently transfer data locally via socket instead of through the network stack (the short-circuit configuration parameter has to be set to enable that connectivity), as well as to enable snappy compression of HBase files.

It was also important to note that since HBase stores the name of family columns and their qualifiers in addition to the row keys and the key values, it caused the predicted storage to double and almost triple in size of table space on each of the six nodes (the name of the columns were shortened but kept as close as possible to the ADT and DAD tables and this reduced a minimal occupancy of the table space). Additionally, HBase has several qualifiers and key stores influencing the Reduce part. And, at each of the iterations, the key store had to be re-indexed. Therefore, to generate and add to the existing dataset and distributed database, only 50 million were ran iteratively because if indexing failed, it was easier to clean and restart compared to 100-200 million rows (Table 11). However, HBase does have an index, which the platform used as a patient encounter as Big Integer to query the replicated without limitation of its numerical index field. Otherwise the replicated data with no indexing meant that no part of the entire database could be queried.

Table 11. Operational experiences, persistent issues and overall limitations of tested big data technologies and components that impacted the Big Data Analytics (BDA) platform.

Technology and Component	Brief Experience	Issue and Limitation	Impact to Platform
Hadoop Distributed Filing System (HDFS)	-Each node requires configuration and monitoring -Distributed Filing system is unbalanced and local disks will not fail over via Hadoop -if the local disks differed in size 500GB versus 1TB versus 2TB, once full capacity hit, HDFS would crash	-Files need to be distributed with relatively the same -The local disk reaches max 90% and should re-distribute via inherent HDFS's processes but does not quickly re-balance -WestGrid's failover to disks doesn't work because Hadoop is moving the files to re-balance the nodes -no failover for Hadoop from full disks to available (also issue with WestGrid)	-Did not reconfigure more than 6 nodes because very difficult to maintain and ongoing issues, if one unbalanced then it will either drastically slow the Reducer or it will be in quasi un-processing state with constantly moving files -Had to add additional 2-4TB of local disks because issue persisted with running to 3 billion -impact is database ingestion inoperable and need to cleared and restarted. -had to implement large local disks of all the same size to avoid Hadoop HDFS crash
MapReduce	-Map component is extremely fast at 3-12mins for each	-Reducer fails and ingestion needs to be cleared from nodes	-Totally failed ingestion and system inoperable -Index files need to be

	iteration -Reduce can start at 10-40% during the Map and this variation is not controllable -Reducer hits 99% after 12 hours and crashes -Optimized iterations at 50 million rows as it took long to run for 100 million and more difficult to monitor	and module load restarted -Reducer is extremely slow -Reducer is placing the files only on one node	removed from node and restarted to complete the iteration -Extremely slow performance to form the database and requires constant monitoring -Current major limitation and more advanced algorithms and the java coding for MapReduce needs to be further explored, verified and implemented.
HBase	-All five RegionServers need to run in unison otherwise unbalanced HDFS and poor performance -InfiniBand not accessed by RegionServer - HBase qualifiers and key stores influencing the Reduce part at each of the iterations the key store had to be re-indexed	-There is an error message via a customized script when restarting the bulkload to HBase but sometimes the Reducer will place the data on the ones available even though it should stop -RegionServers slow or killed because of lack of connectivity -RegionServers constantly died but resolution was to run compaction after ingestion -HBase cannot re-index data from either HFiles that crashed and didn't complete at any Reducer level, even at 99-100%	-RegionServers are required for the ingestion to form the database, without them it is not operational -Ongoing monitoring and log checking if the RegionServers are down or not connected to InfiniBand -The script to prompt user that 5 RegionServers dead provided better usability as finding the log files is tedious work and time consuming -Run compaction with shell script after each iteration to HBase -ran only 50 million because if indexing failed, easier to clean and restart than 100 million rows -if index failed or space on local disk maximized, had to re-run all HFiles to bulkload via MapReduce to HBase again, each iteration was set to start +1 from the last and this setting was manually done

ZooKeeper and Yarn	-ZooKeeper services need to be ongoing and configuration done for InfiniBand in its relation to RegionServers	-ZooKeeper not allocating and slow	Extreme slow performance when ZooKeeper services are not running properly but additional configuration minimized this limitation with few issues thereafter
Phoenix	-Length of columns too long and need to be matching in the schema and on the distributed nodes	-Extremely slow performance in ingesting the files of column names are more than 12 characters -Queries will return an error if the column names do not match	-Maintain a database schema with current names in a file on the nodes such that if the files ingested don't match it will show error and to verify while running queries -Zero times this occurred while ingesting files but many times at first when running queries
Spark	-Relies on Yarn -Advanced Programming	-Yarn not allocating and slow -Java code (Scala) required	-Potential slow performance if not coded correctly -Check online code sources
Zeppelin	-Delay to run the SQL-like script in its initial additional ingestion of file -SQL-like code is more complicated than traditional SQL	-Delay to start testing or running queries	-30minute delay before running queries which takes the same amount of time as with Jupyter -No fix to this issue
Jupyter	-Need to perform some Java coding to produce graphs	-No graphs produced and no buttons on interface available like Zeppelin	-Once the Java is established it has high usability an excellent performance
Drill	-Can only plugin one SQL code at a time -Relies on ZooKeeper	-Poor usability -ZooKeeper not allocating and slow	-Extremely fast but poor usability -It was recently developed so newer version will have a better interface or at least some integration to other interface engines

There were 22 SQL queries tests for querying reports, instances, and frequencies in the ADT/DAD data over the 50 million to 1-3 billion rows. Ten queries were categorized as simple while others were complex; these included more than three columns and three primary keys

across the 90 possible columns. All queries simple (linear) and complicated (exponential and recursive) were less than two seconds for one billion and almost the same for three billion when the nodes were, eventually, balanced by Hadoop; however, some queries were more than three seconds and less than 4 seconds for three billion with unbalanced nodes (Table 12). No significant differences between simple and complex, and possible two second increase when nodes are unbalanced. It was checked several times to run query but caching did not influence the query times. Below is an example of a simple query of ADT and DAD combined for "medical services with unit transfer occurrence" (with a run time of a few seconds):

SELECT DAD.EPISODE_Duration, DAD.ANESTHETIC_Code, DAD.Intervention_Location, DAD.MedicalServices, DAD.TransferLocation
FROM DAD
Group BY DAD.AGE;

Additionally, below is a complex query, as more than one "Group By" represented an exponential or two-fold process to complete the query for "frequency of ambulance and diagnosis with Length of Hospital Stay (LOS)":

SELECT by DAD.DIAGNOSIS_CODE, DAD.AdmitByAmbulance, DAD, DischargeDisposition, DAD.InterventionOccurence, DAD.DAD, MedicalServices, Count(DAD.DIAGNOSIS_CODE) AS Frequency, DAD.LOS
FROM DAD
GROUP BY DAD.GENDER, DAD.AGE;

There was no significant difference in the performance of simple versus complex queries (Table 12). The performance speed, even at one to three billion rows for complex queries, was extremely fast compared to the 50 million rows queried. It was projected that Spark at 6 billion (from preliminary tests) would show more efficient queries at larger volumes and thus it its performance curve less steep compared to Drill's performance towards 6 Billion (Figure 20).

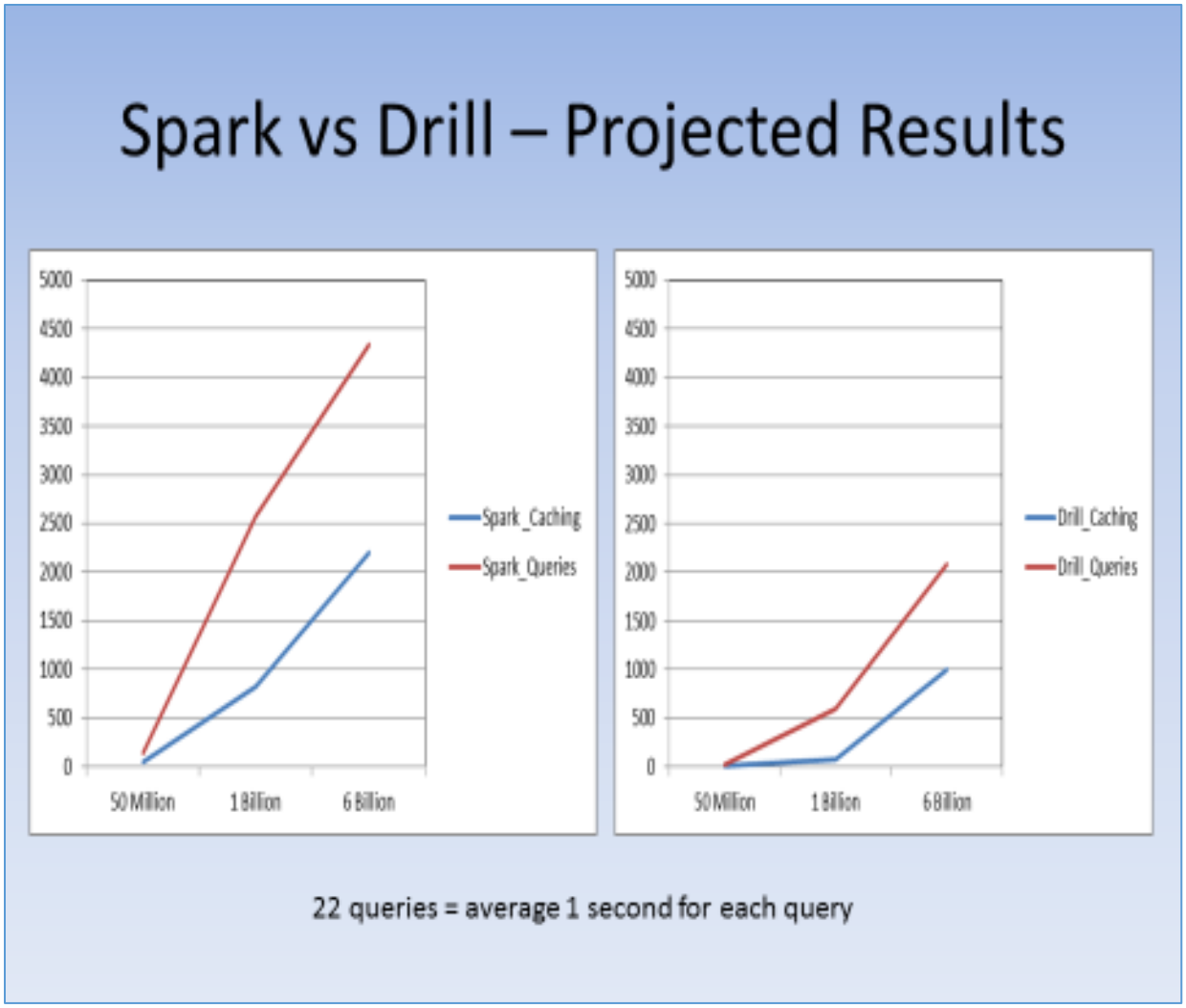

Figure 20. Projected results for 6 Billion records for Spark and Drill.

It did require months of preparation to get to the task of testing the platform with the queries. Health data that was involved with hospital outcomes and clinical reporting was combined to form a database and distributed over nodes as one large file, up to 30TB for HBase. All the pertinent data fields and much more were used. This is important since, for example, VIHA has incorporated SNOMED in its ADT system, as an alternative field to ICD-CA codes that is in their data acquisition from clinical applications, but has not yet integrated it into their data warehouse.

Table 12. One-time duration (seconds) of performance of queries run by Apache Phoenix over 50 million (M), 1 Billion (B) and 3 Billion (B) with unbalanced* and balanced** across the Hadoop cluster with HBase NoSQL datasets.

Description	Type	Apache Phoenix SQL-like Query	Output Efficiency (OE) 50M (seconds)	Output Efficiency (OE) 1B (seconds)	Output Efficiency (OE) 3B *unbalanced (seconds)	Output Efficiency (OE) 3B **balanced (seconds)
#1. Basic selection of encounter data	Simple	*select * from DADS1 where EncounterID<10010 and EncounterID>10004;*	1.84	1.87	3.87	3.05
#2. Basic selection of encounter data based on admitted via ambulance	Simple	*select * from DADS1 where EncounterID<10010 and EncounterID>10004 and ADMIT_BY_AMBULANCE='C';*	1.83	1.77	3.65	1.65
#3. Frequency of Diagnosis (Dx) Code with LOS	Simple	*select Diagnosis_Code, COUNT(Diagnosis_Code), AVG(LOS) from DADS1 where EncounterID<1000100 and EncounterID>1000000 GROUP BY Diagnosis_Code;*	1.14	1.47	3.11	2.11
#4. Frequency of Diagnosis (Dx) Code with Diagnosis and LOS	Simple	*select Diagnosis_Code, COUNT(Diagnosis_Code) as Frequency, LOS from DADS1 where EncounterID<1000100 and EncounterID>1000000 GROUP BY Diagnosis_Code,LOS;*	1.64	1.68	3.32	2.32
#5. Diagnosis Code with Discharge date and Discharge time	Simple	*select Diagnosis_Code, Discharge_Date, Discharge_Time from DADS1 where EncounterID<1000010 and EncounterID>1000005;*	0.83	0.77	3.02	1.02

#6. Diagnosis Code with Unit Transfer Occurrence	Simple	*select Diagnosis_Code, COUNT(Diagnosis_Code), AVG(Unit_Transfer_Occurrence) from DADS1 where EncounterID<1000100 and EncounterID>1000050 GROUP BY Diagnosis_Code;*	1.15	1.22	3.67	1.67
#7. Diagnosis Code with Location building, Location Unit, Location Room, Location Bed, Discharge Disposition	Simple	*select Diagnosis_Code, Location_Building, Location_unit, Location_Room, Location_Bed, Discharge_Disposition from DADS1 where EncounterID<1000010 and EncounterID>1000000;*	1.16	0.84	3.23	0.98
#8. Diagnosis Code with Encounter Type and LOS	Simple	*select Diagnosis_Code,Encounter_Type,LOS from DADS1 where EncounterID<1000010 and EncounterID>1000000;*	0.69	0.68	3.01	0.98
#9. Diagnosis Code with Medical Services and LOS	Simple	*select Diagnosis_Code, Medical_Services, LOS from DADS1 where EncounterID<1000010 and EncounterID>1000000;*	0.44	0.38	3.00	1.02
#10. Provider Service with Diagnosis codes	Simple	*select Diagnosis_Code,Provider_Service from DADS1 where EncounterID<1000010 and EncounterID>1000000;*	1.35	1.44	3.52	1.92

#11. Highest LOS for MRNs with Admit date	Simple	*select LOS, MRN, Admission_Date from DADS1 where EncounterID<1000100 and EncounterID>1000050 GROUP BY LOS, MRN, Admission_Date ORDER BY LOS DESC;*	1.45	1.46	3.62	1.62
#12. Frequency (or number) of Admit_category with Discharge_Date	Simple	*select Admit_Category, COUNT(Admit_Category) as Frequency, Discharge_Date from DADS1 where EncounterID<1000100 and EncounterID>1000050 GROUP BY Admit_Category, Discharge_Date;*	1.86	1.76	3.54	1.89
#13. Admitted by Ambulance, Interventions, and Medical Services with Diagnosis	Complex	*select Gender, COUNT(Admit_by_Ambulance), COUNT(Discharge_Disposition), COUNT(Interven_Occurrence), COUNT(Medical_Services), COUNT(Diagnosis_Code), MAX(LOS) from DADS1 where EncounterID<1000010 and EncounterID>1000000 GROUP BY Gender;*	1.79	1.87	3.67	1.89
#14. Intervention and Location with Admit and Location	Complex	*select Interven_Occurrence, Interven_Episode_St_Date, Interven_Location, Interven_Episode_Start_Date, Interven_Attribute_Location, Admission_Time, Location_Unit, Location_Bed, Location_Building from DADS1 where EncounterID<1000010 and EncounterID>1000000;*	1.64	1.75	3.03	1.87

#15. Medical Services with Unit Transfer Occurrence	Complex	*select Count(Episode_Duration), Count(Anesthetistic_Technique), Count(Interven_Location), Count(Medical_Services), Count(Unit_Transfer_Occurrence) from DADS1 where EncounterID<1000010 and EncounterID>1000000 Group BY age;*	1.83	1.75	3.47	1.92
#16. Admit Category and Discharge with Transfer	Complex	*select LOS, Count(Discharge_Disposition), Count(Most_Responsible_Site), Max(Transfer_In_Date), Min(Transfer_Out_Date), Max(Transfer_Hours), Max(Days_In_Unit), Count(Patient_Service), Max(Patient_Service_Occurrence) from DADS1 where EncounterID<1000010 and EncounterID>1000000 GROUP BY LOS;*	1.64	1.75	3.61	1.75
#17. Encounter, Discharge and Transfer	Complex	*select Diagnosis_Code, Encounter_Type, LOS, Admit_Category, Discharge_Date, Discharge_Time, Location_Building, Location_Unit, Location_Bed from DADS1 where EncounterID<1000010 and EncounterID>1000000 ORDER BY Diagnosis_Code DESC;*	1.85	1.76	3.56	1.53

#18. Medical Services and Days in Unit	Complex	*select Patient_Service_Days, Patient_Service_Occurrence, Transfer_In_Date, Transfer_Out_Date, Days_In_Unit, Medical_Services, Location_Unit from DADS1 where EncounterID<1000010 and EncounterID>1000000;*	1.82	1.90	3.72	2.34
#19. Admission, Transfer with Intervention and Encounter	Complex	*select LOS, Count(MRN), Count(Admission_Date), Count(Admission_Time), Max(Institute_From), Count(Admit_Category), Count(Encounter_Type), Count(Entry_Code), Count(Diagnosis_Code), Max(Interven_Episode_St_Date), Count(Interven_Attribute_Extent) from DADS1 where EncounterID<1000010 and EncounterID>1000000 and LOS BETWEEN 0 AND 9999 GROUP BY LOS ORDER BY LOS DESC;*	1.97	1.88	3.82	3.02
#20. Frequency (or number) of Admit_category with Patient Service	Complex	*select Admit_Category, AVG(Patient_Services_Occurrence), COUNT(Patient_Service_Type), MAX(Transfer_In_Date), MAX(Transfer_Out_Date), Count(Transfer_Nursing_Unit), Count(Service_Nursing_Area), Count(Medical_Services), Count(Encounter_Type), Count(Diagnosis_Type),*	1.85	1.79	3.86	2.61

		count(Location_Unit), count(Provider_Service) from DADS1 where EncounterID<1000010 and EncounterID>1000000 GROUP BY Admit_Category, Discharge_Date;				
#21. Provider Occurrence with Nursing	Complex	*select Provider_Service, Provider_Type, Diagnosis_Code, Provider_Occurrence, Transfer_Nursing_Unit, Medical_Services from DADS1 where EncounterID<1000010 and EncounterID>1000000;*	1.62	1.65	3.78	1.70
#22. Provider with Diagnosis and Intervention	Complex	*select Provider_Service, Provider_Type, Provider_Occurrence, Diagnosis_Code, Diagnosis_Type, Medical_Services, Unit_Transfer_Occurrence, Interven_Code, Interven_Occurrence, Interven_Provider_Service, Interven_Episode_St_Date, Interven_Attribute_Extent from DADS1 where EncounterID<1000010 and EncounterID>1000000;*	1.85	1.75	3.86	1.71

v. Hadoop-Spark-Drill – Technical Implementation and Challenges

From a technical perspective, very few issues were faced during software installation of Spark and Drill. Besides the lack of documentation, a major issue with Hadoop's ecosystem of packages was that many of them failed to include the dependencies that are local to the system, like with ZooKeeper for Spark and the version of Hadoop required with version of Python (Table 11). To establish the BDA framework with different big data technologies, therefore, additional configurations had to be made in order to customize the platform without failed services and ongoing running of Hadoop, which was required to run ingestions and query. Additionally, for improved efficiency, the platform used native libraries because Hadoop needs to be recompiled from source (Change et al., 2014; Dhyani & Barthwal, 2014; Dunning et al. 2016; Jurney, 2013; Lai et al., 2014; Sakr & Elgammal 2016). And it was found that many packages do not offer a profile that allowed them to be used, as local Hadoop even avoids downloading components. Moreover, given that the end users were allowed to run their Hadoop module on WestGrid's clusters. Therefore, it was important that the configuration directories had most of the Hadoop's ecosystem of packages built in to minimize reliance on them. Unfortunately, the configuration of Hadoop was not propagated properly to all the remote services running on the nodes, which caused the default configuration on those nodes to be used.

Another major challenge faced was that Apache Drill did not have any option to force it to bind to a specific network interface. As a result, its *Drillbits* started used the network management instead of the InfiniBand network. Zeppelin used *databricks* from Spark to ingest the file that was run over the interface (Figure 21). Similarly, it also intermittently occurred when initializing Zeppelin and this added an additional 30 minutes to the ingestion and query of 1-3 billion rows. In addition to having to move from the 0.5.0 version (as it did not support Spark 1.5.2) to the latest snapshot (0.6.0), the Zeppelin's *pyspark* timed out the first time *%pyspark* was run. In fact, it was found that the user has to wait for a couple of seconds and execute the cell one more time to get pyspark working. Another challenge with Zeppelin, even though it offers a pull-down list of notebooks (Figure 22), was that the notebook could only be exported as a JSON file. Saving the notebook as an html page produced empty webpages and that exhibited very poor usability. Therefore, the Zeppelin notebooks can only be open under Zeppelin. There were, however, some off-the-shelf buttons to plot data as a data visualization tool for Zeppelin (Figure 23).

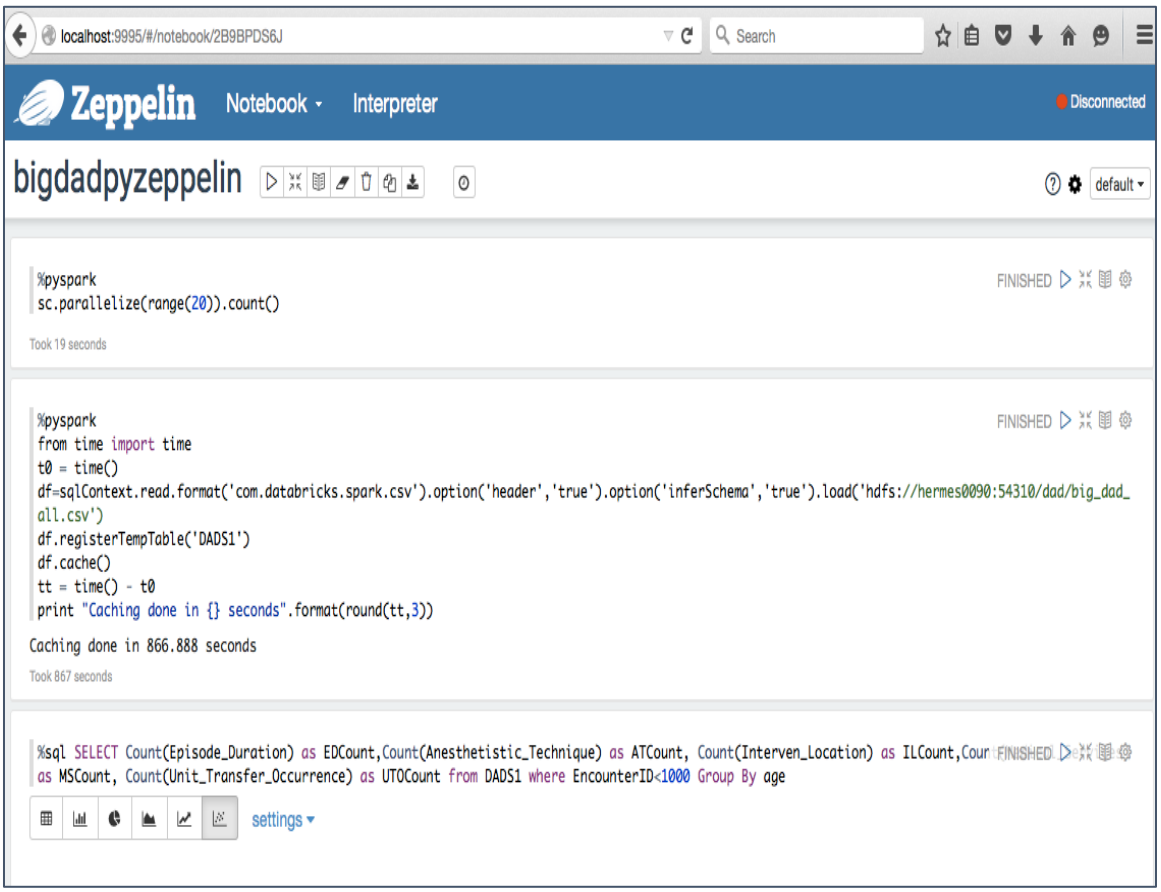

Figure 21. Ingestion script with databricks of Apache Spark over Zeppelin.

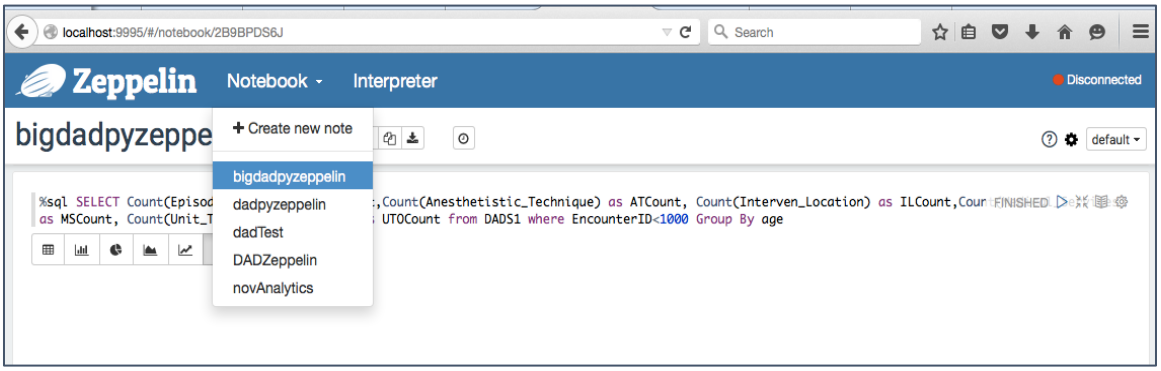

Figure 22. Creation of notebook interface with Spark.

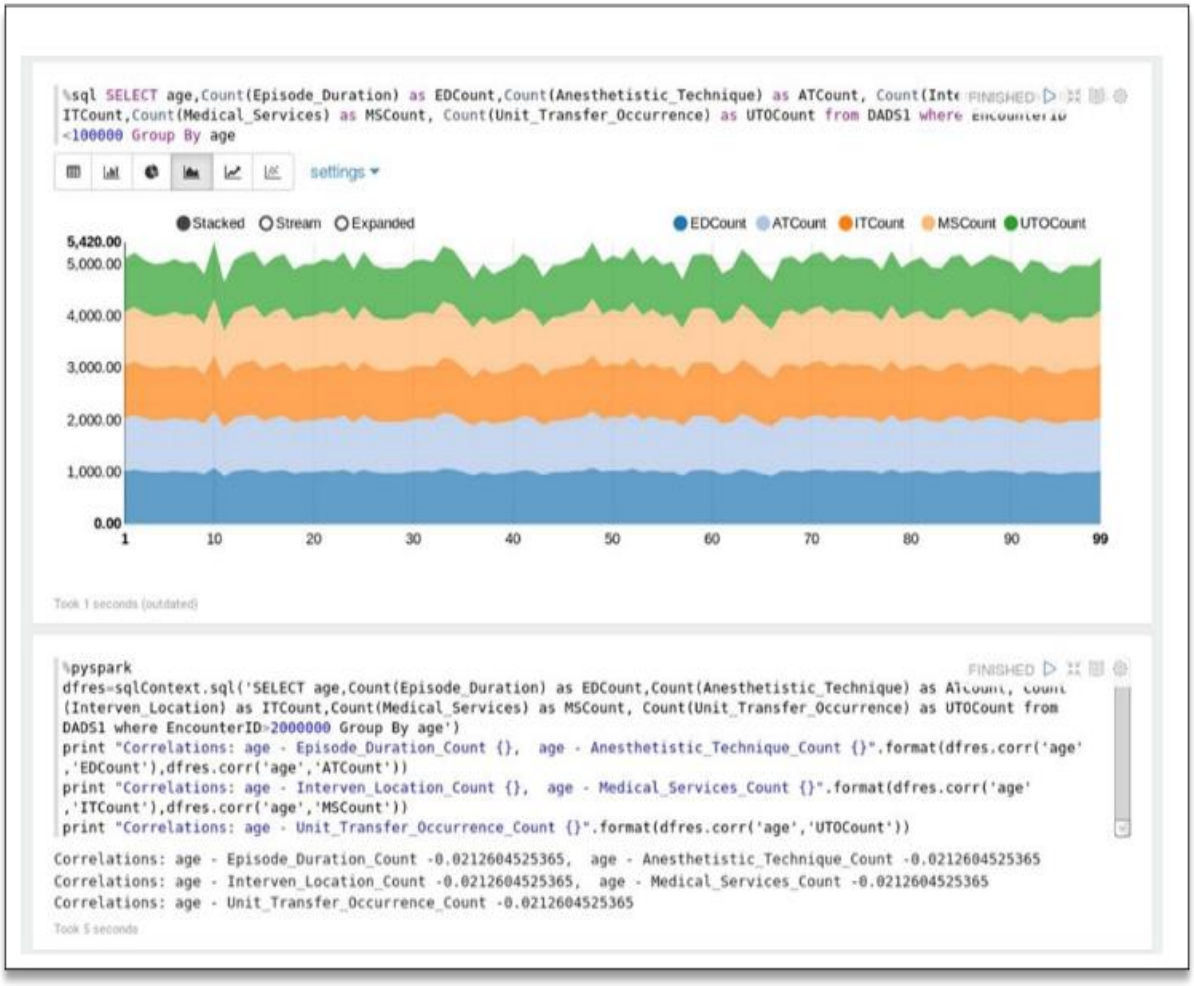

Figure 23. Visualization and simple correlation analytics within Zeppelin using *Pyspark*.

vi. Hadoop-Spark-Drill – Efficiencies and Outputs

Four *.csv* files were ingested to HDFS: 1) one with 10 records was used for quick testing; 2) one with 50 million records was used for the actual benchmarking; and, 3) another two with one and three billion records. The results are show in Table 13. It was set as ingestion and query efficiency as the same calculation as for HBase (refer to 3.0 Material and Methods section).

Table 13. Hadoop Ingestion Time.

Data Size	50 Million records (23 GB)	1 Billion records (451 GB)	3 Billion records (10 TB)
Ingestion Time	~6 min	~2 h 5 min	~5 h 2 min

vii. Hadoop-Spark-Drill – SQL Queries

The queries were run on Zeppelin, spark-shell, and *Pyspark*; all took approximately the same amount of time (Table 14). This was not surprising, as they all rely on Spark SQL. Therefore, it was only reported as a single time for all three.

Table 14. SQL Querying Time for Spark and Drill.

SQL Engine	Spark SQL (seconds)	Drill SQL (seconds)
50 Million Records	194.4	25.5
1 Billion Records	3409.8	674.3
3 Billion Records	5213.3	1543.2
Query Efficiency	1.14-1.52	0.76-0.84

Spark was configured to run on a Yarn-client with 10 executors, four cores with 8 GB of RAM each; therefore, each node had two executors with a total of eight cores and 16 GB memory (tests using one executor with 16 GB and eight cores on each node was slightly less efficient). The Zeppelin code snippet used was:

```
%pyspark
from time import time t0 = time()
df=sqlContext.read.format('com.databricks.spark.csv').option('header','true').option
('inferSchema','true').load('hdfs://h ermes0090:54310/dad/big_dad_all.csv')
df.registerTempTable('DADS1')
df.cache()

%sql SELECT Count(Episode_Duration) as EDCount,
Count(Anesthetistic_Technique) as ATCount, Count(Interven_Location) as
ILCount, Count(Medical_Services) as MSCount,
Count(Unit_Transfer_Occurrence) as UTOCount from DADS1 where
EncounterID<1000 Group By age
```

Drill was configured to run the *Drillbits* on the Hadoop DataNodes (with a single Drillbit on each node). Each Drillbit was configured with a VM heap of 4 GB and a maximum direct memory of 16 GB (standard configuration). The queries were run from both the web interface of one of the *Drillbits* and the *sqlline* command line. Both gave the same timing. The Drill code was:

```
SELECT Count(Episode_Duration) as EDCount, Count(Anesthetistic_Technique) as
ATCount, Count(Interven_Location) as ILCount, Count(Medical_Services) as
MSCount, Count(Unit_Transfer_Occurrence) as UTOCount from
dfs.`/dad/big_dad_all.csv` where EncounterID<1000
Group By age
```

The results clearly showed that Drill was five to 7.6 faster than Spark, which indeed justified its use as a low latency query engine tool (Table 14). The Phoenix and HBase benchmarking reported earlier (Chrimes et al., 2016) showed that HBase outperformed all of them and the HBase queries were all within a second or two. This was an astonishing performance and HBase was used, when possible, as the underlying data source for the HBDA platform.

It was surprising that the query efficiency (*QE*) of Drill was only 76%. It was believed at the time of running both engines that this was due to a lack of binding to the InfiniBand interface. The Drill Developers were contacted about this and there was work with them to debug for an eventual break fix. Drill's query processes, however, were still not as efficient as that of Spark with increased database size to ingest.

4.3 Usability, Simple Analytics, and Visualizations

The results showed that the ingestion time of one billion records took circa two hours via Apache Spark. Apache Drill outperformed Spark/Zeppelin and Spark/Jupyter. However, Drill was restricted to running more simplified queries, and was very limited in its visualizations that exhibited poor usability for healthcare. Zeppelin, running on Spark, showed ease-of-use interactions for health applications, but it lacked the flexibility of its interface tools and required extra setup time and 30-minute delay before running queries (Table 11). Jupyter on Spark offered high performance stacks not only over the BDA platform but also in unison (Table 11), running all queries simultaneously with high usability for a variety of reporting requirements by providers and health professionals.

Being able to perform low latency SQL queries on a data source is not enough for healthcare providers, clinicians, and practitioners. Interacting with and exploring the data through different analytics algorithms and visualizations is usually required to get the data's full value. A variety of functionalities and tools for expressing data was an essential quality to test over the platform.

Drill did perform well compared to Spark, but it did not offer any tools or libraries for taking the query results further (Table 11). That is, Drill proved to have higher performance than Spark but its interface had less functionalities. Moreover, algorithms (as simple as correlations between different columns) were time-demanding if not impossible to express as SQL statements. Zeppelin, on the other hand, offered the ability to develop the code, generate the mark-down text, and produced excellent canned graphs to plot the patient data (Figures 21-23 shows the interactive development).

Combined with the richness of Spark and pyspark, Zeppelin provided a canned visualization platform with graphing icons, as shown in Figure 23. The plots under Zeppelin, however, are restricted to the results/tables obtained from the SQL statements. Furthermore, the results that were produced directly from the Spark SQL context did not have any visualization options in Zeppelin. Generating results from queries via Zeppelin took much longer (over 30 minutes). Establishing the platform to run queries on the interface and generate results via Zeppelin took longer than Jupyter.

With Jupyter, more configurations with the data queries were tested. It exhibited similar code to ingest the file (Figure 24), same Spark databricks initialized in the interface (Figure 25), and its SQL to query (Figure 26) as Zeppelin, but at the expense of writing the visualization code, using the *matlplotlib* Python package in addition to other powerful tools, such as Pandas, i.e., a powerful Python data analysis toolkit (Figure 27). The local host was added to Hermes node to access Jupyter via the BDA platform to compensate for the lack of visualization options via the Zeppelin interface. Figure 27 shows a small snippet from the output of a Jupyter/Spark interaction that uses both matlplotlib and Java's Pandas.

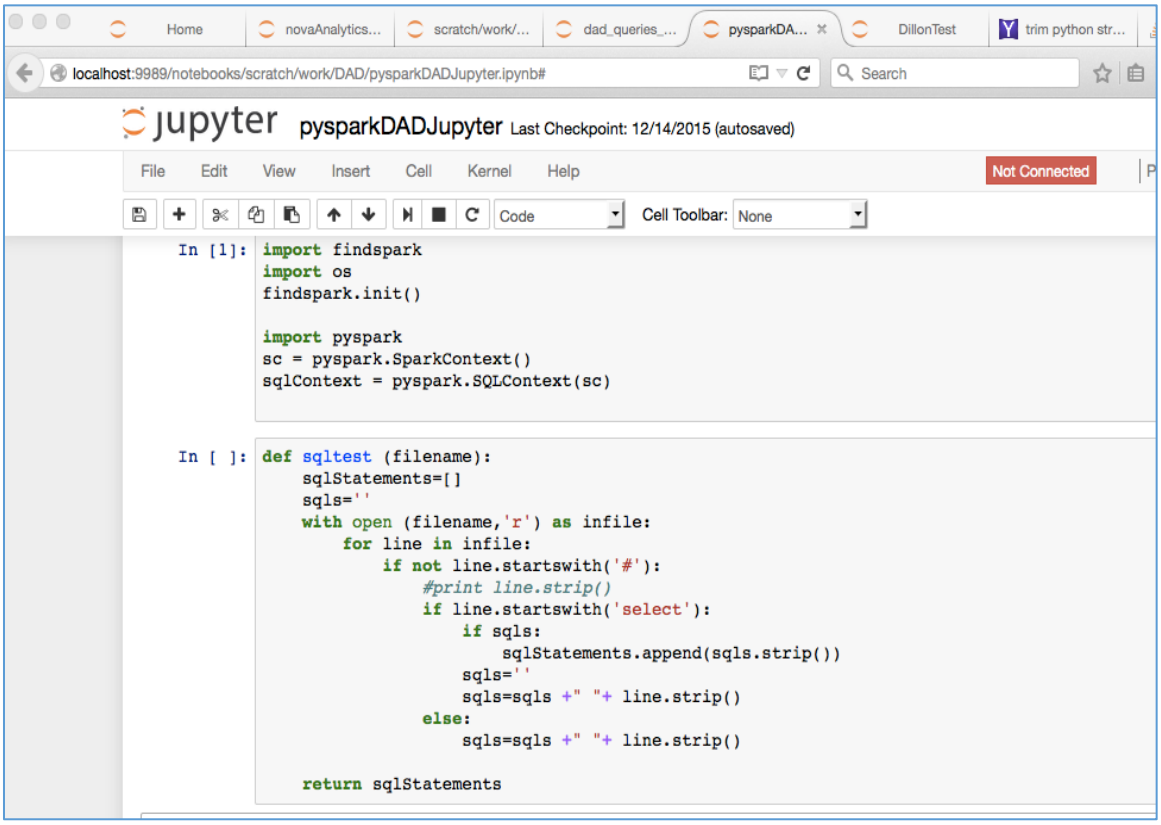

Figure 24. Spark with Jupyter and loading large data file in the Hadoop cluster before SQL was placed.

```
In [2]: from time import time
        ti = time()
        df=sqlContext.read.format('com.databricks.spark.csv').option('header','true').option('inferSchem
        df.registerTempTable('DADS1')
        for sqlstatement in sqltest ('dad_queries.sql'):
            t0 = time()
            sqlContext.sql(sqlstatement).show()
            tt = time() - t0
            print "Query performed in {} seconds".format(round(tt,3))
        #dfres=sqlContext.sql('SELECT Count(Episode_Duration),Count(Anesthetistic_Technique), Count(Int
        #dfres.collect()
        #dfres.show()

        tt = time() - ti
        print "Query performed in {} seconds".format(round(tt,3))

+---+---+---+---+---+
|_c0|_c1|_c2|_c3|_c4|
+---+---+---+---+---+
|  8|  8|  8|  8|  8|
```

Figure 25. Spark with code to import file and formatted via *databricks* from flat file (*.csv*).

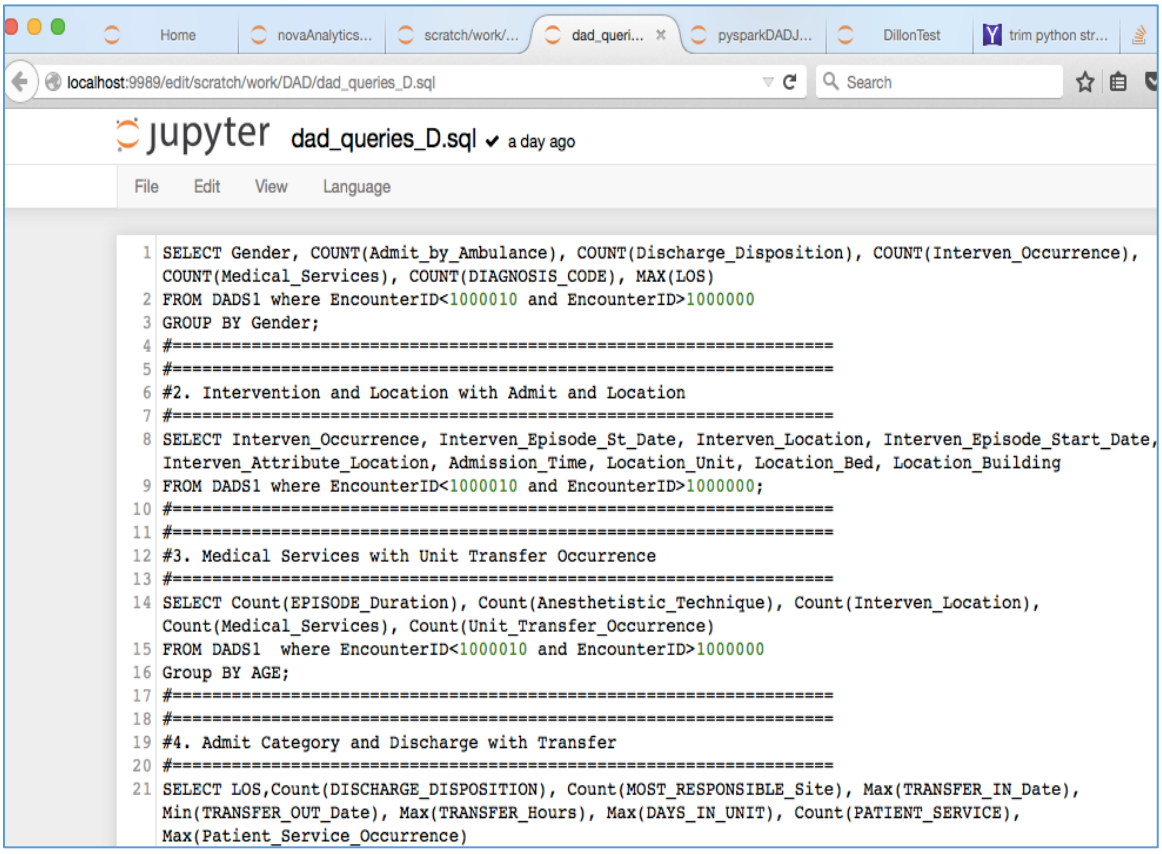

Figure 26. Spark with Jupyter and SQL-like script to run all queries in sequence and simultaneously.

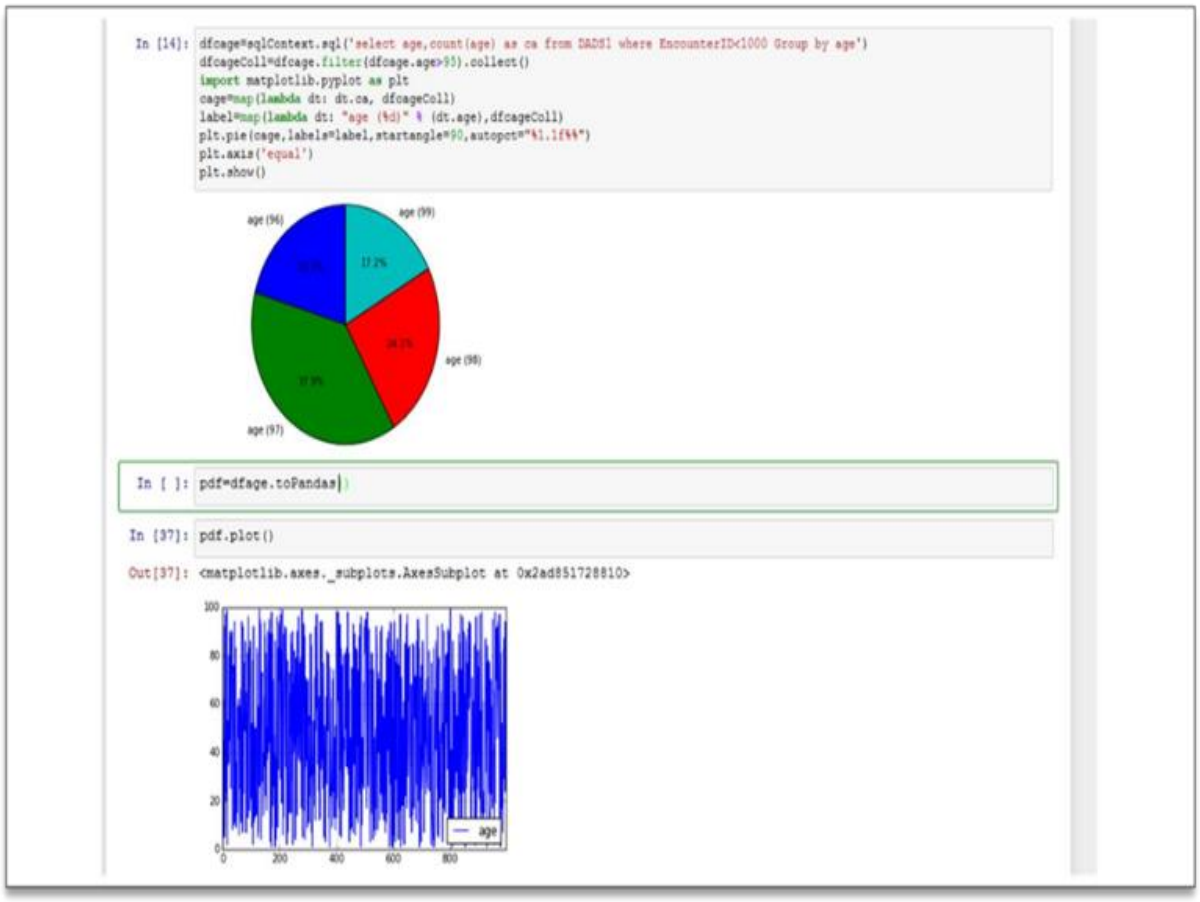

Figure 27. Example of simple Jupyter/Spark interaction on its web-based interface including data visualizations.

Usability of the platform did validate the proof-of-concept of querying patient data with high performance in generating results and visualization over the interface. Performance to generate results had the same number of sequence steps for the end users as HBase, Spark and Drill. Running Apache Spark took the same amount of time for queries on Zeppelin and Jupyter; however, the initial setup (to start the queries) took much longer via Zeppelin than Jupyter. Drill seemed to have simplified steps to the setup interface compared to Spark and took significantly less time; therefore, it appeared to have better usability. Nevertheless, Jupyter supplied more visualization defaults and customization than Drill for its distributed mode (Figure 28) and its interface to run the query (Figure 29) where severely lacking any visualization.

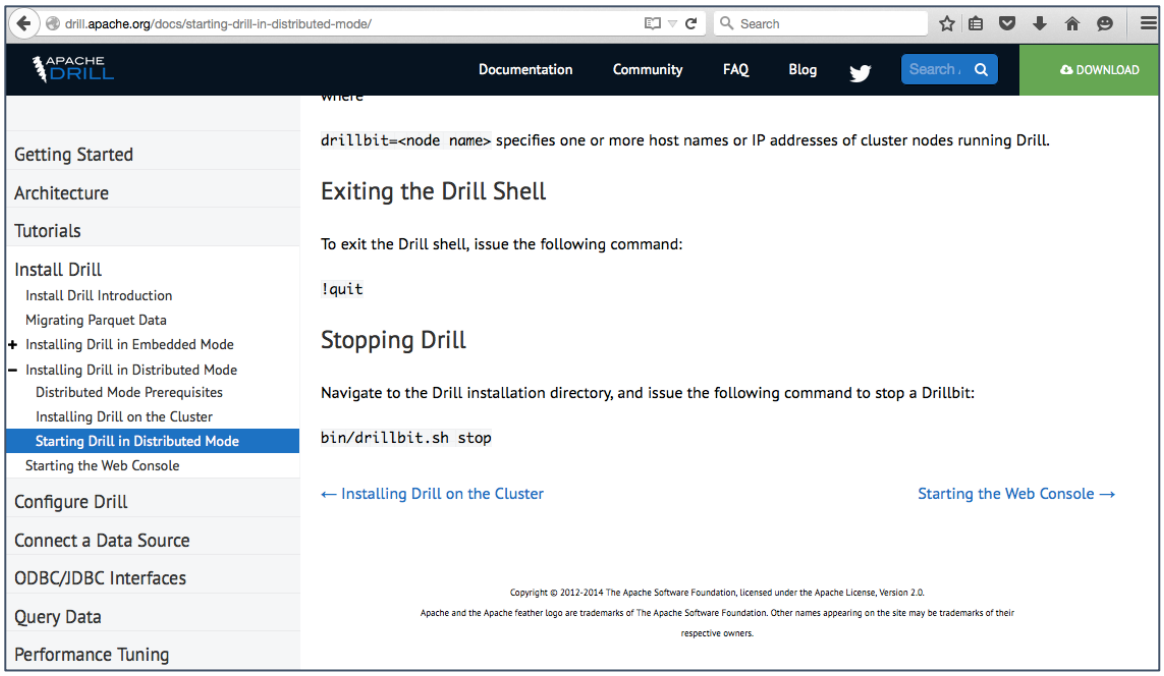

Figure 28. Starting Drill in Distributed Mode as default to its application.

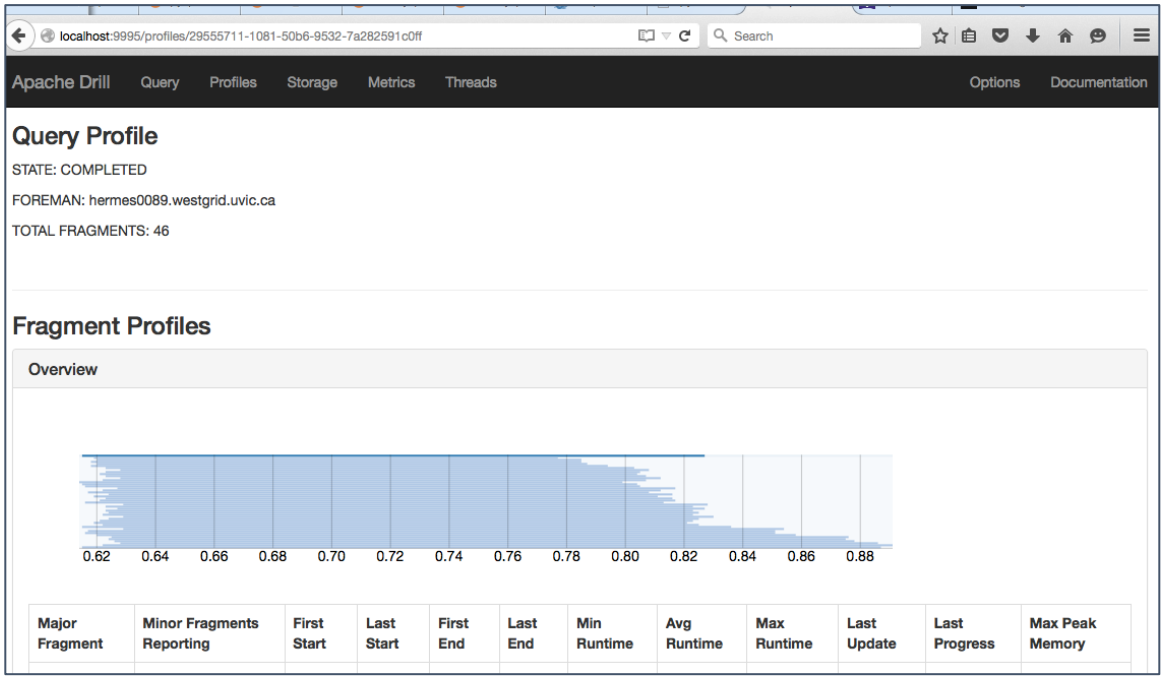

Figure 29. Drill interface customized using the distributed mode of Drill with local host and running queries over WestGrid and Hadoop**.**

5. Discussion

5.1 Big Data in Healthcare and the Importance of MapReduce

The ultimate goal of the study was to test the performance of the Big Data computing framework and its technical specifications over a platform against all challenges specific to its application in healthcare. This goal was accomplished by combining ADT and DAD data through ingestions over the Hadoop HDFS and the MapReduce programming framework. The resulting dataset comprised 90 columns and 3 billion rows, creating a functioning (somewhat dynamic in its modifications on the fly) interactive platform. It was not established in order to discuss the data simulation and modeling but rather to test the end result over a functional platform was usability testing of Big Data tools in healthcare in much greater detail than had ever been done before. Clinical reports from hospital system are usually produced in batches in an automated system in which they are solely derived with known columns and with abstracted data elements for already-established data standards (Madsen, 2014). High performance over the BDA platform was verified with query times of less than four seconds for three billion patient records (regardless of complexity), showing that challenges of aggregation, maintenance, integration, data analysis, and interpretative value can be overcome by BDA platforms. The next major step would be to demonstrate the benefits of BDA with real patient data. However, such a demonstration would involve several levels of analysis to consider. For example, McCormick, Ferrell, Karr, and Ryan (2014) mention that observational healthcare data sets, such as EHR databases, provides longitudinal clinical information at the individual level. Therefore, since this was a proof-of-concept study of a HBDA platform in simulated performance of queries, further investigation is required to verify its results (fully test hypotheses) in real patient data queries. More validation of its ability to hold the health informatics data integrity and correctness is absolutely necessary when going towards using real data.

There are analytical challenges in many Canadian healthcare systems because of separated silos of aggregations, poorly integrated distribution and extractions, and independent origins of the medical information does not allow for real-time analytics to be implemented at large scales (Dufresne, Jeram, & Pelletier, 2014). Besides, there are pools of patient data that cannot be fully integrated with other hospital data because of complex and unique variables that include "(1) information used; (2) preference of data entry (3) services on different objects; (4) change of health regulations; (5) different supporting plans or sources; and (6) varying definition of database field names in different database systems" (Yang et al., 2013). However, the Hadoop/MapReduce framework in this study allowed not only for replication of the data to another destination or platform, which reduces architectural and resource pressure for high-performance production systems, but also allowed for combining data from different databases at each iteration continuously. Additionally, Song and Ryu (2015) showed similar results in their implemented BDA framework in a healthcare system in South Korea that not all data was obtained but large volumes were immediately reached in its storage continuously. Therefore, this study's high-performance outcome was operationally found and did take the complexity of systems and data in healthcare into consideration; however, as a first step, it only documented the experiment of establishing a platform with simulated patient

data with continued use over several months in a few years. In a real setting, the platform will be used indefinitely, therefore, constant maintenance would be astronomically time consuming and expensive. Hence, this study can discuss nuances in healthcare and its challenges, and it can confirm that the objectives were achieved and that the maintenance of a productive HBDA platform remains time-consuming and requires constant monitoring, especially for data ingested into the system via Hadoop/MapReduce that needs to coincide with metadata of health informatics.

The working hypothesis was accepted that larger volumes entail a slight linear increase in query times. However, there were differences in the performance of the ingestions using Hadoop/MapReduce that limited data aggregation and maintenance. This study did not have the resources to implement a streaming process or to expand from the six nodes to one hundred (as originally planned). At the outset there were too many configurations to change and monitor over WestGrid's architecture. Therefore, it was decided to use the Hadoop platform with MapReduce, as ingestions in iteration of HFlies to HBase, followed by simplified ingestion into Spark and Drill, and to recommend such a platform towards real-time healthcare application.

There are several reconfigurations of HBase and Hadoop that can increase the ingestion time or the time to distribute the data; these involve changes in the site, yarn, and RegionServer XMLs. MapReduce was to blame for the time needed to ingest 50 million rows varied widely (from 2 to 12 hours). The corresponding Java coding, Java Virtual Machines (JVMs), and Java services were a performance bottleneck, which is common on most platforms (Maier, 2013; Sakr & Elgammal 2016). The lack of significant differences in the performance or times of simple versus complex queries on the six nodes of the distributed database of one and three billion rows showed the power of Phoenix co-processors in running the SQL queries in parallel on all RegionServers. Moreover, there were differences in the performance and usability among the Apache Phoenix, Spark, and Drill applications, showing advantages and disadvantages of these Big Data technologies in dealing with analytical and interpretive challenges. Clearly, this study showed that Drill, a software addition to Hadoop developed in 2015 (Dunning et al., 2016), significantly outperformed the others in speed. Drill outperformed Spark and Phoenix in HBase processes before queries were generated. However, its interface lacked any functionality to customize and mine the data, which is what health professionals require (because of the complexity of the data and what its clinical reporting should reveal). This poor usability contradicts the recommendations by Scott (2015) in that in most cases Drill should be used instead of Spark. However, this study provided new insight into more customized Java coding with the combination of Jupyter and Spark that enhanced the platform's dynamic interactivity with end user. It is; therefore, recommend that Spark with Jupyter be used with scripted coding to run ingestion and queries simultaneously in further investigation with real data.

Part of the comparison between Drill and Spark involved usability testing of two separate tools over the BDA platform. Drill had on Drill to test usability but Spark had other interfaces (like Jupyter and Spark), which no other studies have produced results on. This study found that Spark with Jupyter had better usability, but this finding is tentative as the

clinical reporters and health professionals only viewed the screenshots and were not familiar with any of the Big Data technologies. Even though Drill outperformed Spark, it lacks an interface capable of displaying important clinical data. Furthermore, running modules in sequence from Hadoop to Spark or Drill with web clients is too technical for most end users and would require additional refinements to the interface for producing clinical reports. The code can be placed in both Zeppelin and Jupyter Notebooks to run all queries at once over the database. However, the only stakeholders that would truly benefit from this code change or utilization would be the data warehouse team.

The most impactful technology of the Big Data technical components in this study was MapReduce (and Java code performance therein). MapReduce methodology is inherently complex as it has separate Map and Reduce task and steps in its default programming framework as this study discovered. This study's platform was highly dependent on the efficiency of MapReduce in ingesting files over the six nodes, using this workflow: Input → Map → Copy/Sort → Reduce → Output similar to a study by Chen, Alspaugh and Katz (2012). The Map part of the platform showed high performance but the Reduce took more than tenfold longer to complete its schedule; however, once configurations in Yarn, ZooKeeper, and others the Reducer seemed be optimized with iterations of 50 million rows. According to blogs and technical resolutions involved enabling or disabling services or xml settings over the platform as expected to be carried because the system relied heavily on InfiniBand (IB) bandwidth at low latency over WestGrid nodes. The configurations for the Reduce component are difficult to change because of its license by Google with Apache makes customization difficult and complex (and changes were deemed unnecessary, since queries usually took only three seconds to a minute). Furthermore, there are known issue with the combination of MapReduce to HBase, although studies have shown that additional indexing and reduction processes can be added and/or modified at the reducer with an advanced programming method (Greeshma & Pradeepini 2016; Maier, 2013; Sakr & Elgammal, 2016; Taylor, 2010; Yu et al., 2016). However, a customized reduction at the reducer level of this platform to form the simulated database proved to be difficult to overcome and maintain at less than 3 hours for each of the iterations and only 50 million rows at file size 258GB.

All the MapReduce logs were documented and the processes discussed during tests, which adds to the current challenges of data aggregation and maintenance found by this study. In the first stage, a node executes a Map function on a section of the input data, and in the second stage the Reduce function is initialized (MapReduce, 2010). In a Hadoop cluster, a master node controls a group of slave/worker nodes on which the Map and Reduce functions run in parallel (Karim et al., 2013; Nabavinejad, Goudarzi, & Mozaffari, 2016). If a Map working in a system crashes, then the DataNode is lost (Chen, Alspaugh, Borthakur, & Katz, 2012). The Job Tracker will reallocate the task to another Task tracker based on replicated data available in a block. This study also found that the Reducer combined the entire output file and sent it back to an application; however, Map started much earlier than Reduce, but this study could not identify the cause of this variation. More investigation is required to locate the source of

inconsistency in performance of the reducer for each of the iterations of 50 million rows to ingest.

In this study, the MapReduce bulk ingestion to HBase/Hadoop temporary storage was so high that it maximized the storage of the local disks a few times and the number of ingested rows had to be reduced to 50 million. Also, it was found that both Yarn and ZooKeeper's defaulted XML files required certain time intervals, RAM, or table space to be altered for ideal performance of ingestion, which is supported by other studies (Jurney, 2013; Lai et al., 2014). Adjustments included connectivity to the IB and RegionServers of HBase, which did not always achieve the low latency or high bandwidth originally sought using IB. Other studies found that that settings in the XML for the services and servers with HBase and Hadoop required customization (Chang et al., 2014; Dyhani & Barthwal, 2014; Jurney, 2013). Mehdipour et al. (2016) showed that performance metrics of their FOG engine for BDA over the cloud relied heavily on processing speed, network bandwidth, and data transfer size, and that optimizing bandwidth could result in a fivefold increase in speed.

Greeshma and Pradeepini (2016) implemented a distributed file system that saved Big Data files, and a MapReduce algorithm supported the performance of the analytics on a set of clusters. The platform queries did not rely on MapReduce to perform and only controlled the actual ingestion of files to form the database. Spark and Drill can use some components of MapReduce at the initialization of the queries, but this was minimal and non-reliable. The Hadoop approach used in this study's platform did not consume data on a large scale as a whole but broke the computational data into smaller pieces over a collection of servers and iterations, thereby placing load on an AdminNode (deployment) that was deployed to NameNode and then distributed to DataNode. This supports Greeshma and Pradeepini's (2016) findings that NameNode and AdminNode manage everything with respect to storing, managing, and accessing Big Data files. In contrast, Job Tracker employs MapReduce algorithm implementation (Chen et al., 2012; Chen, Zhang, Guo, Deng, & Guo, 2010). Furthermore, this study did not benchmark the actual performance of the components of the platform, whereas Yu et al. (2016) showed that Mahout's MapReduce architecture was eight to five times slower than other modified MapReduce configurations. Moreover, Greeshma and Pradeepini (2016) showed that adding Job Tracker to the MapReduce framework is useful, since Hadoop does need to be running while doing the queries; it solves the limitations of the RDBMS/Data warehouse by continuously running with low resources over multiple nodes. Unlike the situation in a data warehouse, there is then no need to cut a portion and do the analytics; with these technologies, queries can then be made continuously. Because Hadoop cannot stop running, it is in essence the platform itself and a single point of failure, which is supported findings of broke Hadoop clusters by Rabkin and Katz (2013). That is, this study made it clear that the MapReduce framework played an important role with Hadoop in distributing the data accurately at a productive level.

A large body of research has studied ways of improving storage performance (Ahmad, Lee, Thottethodi, & Vijaykumar, 2012; Moise, Trieu, Bouge, & Antoniu, 2011) or using compression techniques to reduce the volume of intermediate data (Ruan, Zhang, &

Plale, 2013; Xue, Shen, Li, Xu, Zhang, & Shao, 2012). The intermediate data movement time during the shuffle phase has been addressed (e.g., Ahmad, Chakradhar, Raghunathan, & Vijaykumar, 2014; Wang, Xu, Li, & Yu, 2013; Yu, Wang, Que, & Xu, 2015). However, none of these studies has addressed the issue of memory limitation in the Reduce phase, which can lead to abortion of Reduce tasks as reported by Yan, Yin, Lian, et al. (2015), who indicated the importance of memory resources (which is exactly what occurred with ingestions in this study, although the platform did not fail any time queries were run). Greeshma and Pradeepini (2016) did propose a memory cached mechanism to utilize with MapReduce to improve the Reducer, but this was not tested on other software technologies like HBase. Additionally, while saturation of storage IO operations or network bandwidth can lead to performance degradation in the Reduce phase (a problem encountered on Regional Servers), the MapReduce application cannot be killed by the Hadoop framework in such cases (Ahmad et al., 2014; Greeshma & Pradeepini, 2016); its execution continues, although slowly with poor performance, which supports this study's operational findings. However, in the case of an out-of-memory error, the job is killed, since the Reduce phase needs to bring large portions of intermediate data into memory for processing (Nabavinejad et al., 2016). The operational findings showed that if there is not enough memory space left, the Reduce tasks and, consequently, the Reduce phase (at a given percentage) failed, which then caused a job termination by Hadoop. In this respect, there is a major difference between shortage of memory and shortage of other resources (such as disk/network IO or CPU) in MapReduce applications; the latter presents a significant challenge to productive operations. However, if memory becomes the bottleneck, one will face performance degradation even when the job is not killed.

Lack of memory is not the only reason for failures of MapReduce jobs; other factors include disk failure, shortage of disk space, and sockets timeouts (Chen, Lu, & Pattabiraman, 2014; Dinu & Ng, 2012; Faghri et al., 2012; Moise et al., 2011; Zhou, Lou, Zhang, Lin, Lin, & Qin, 2015). Such issues are not the focus of this work, since they are imposed by *external* effects (e.g., by faults such as disk failure or network timeouts, or by inadequate resources, such as disk space); in this study, the main cause of the failure is *internal* to the operation of the application. The study considered external factors as potentially part of the WestGrid ecosystem, but review of the log files suggests that Reduce components failed, as Java exceptions, because of timeout or because the server died nine times out ten more than because of faults in the external ecosystem of WestGrid or unplanned downtime.

While ingestions in this study were extremely fast, the bulkloads of 50 million rows in iteration to one billion were slow and took, collectively, one week. Ingestions to three billion were even slower, but these times are as fast as or faster than current data migrations (of a similar size) estimated at VIHA. More importantly, query times were less than two seconds for all queries, which is significantly faster than current estimated query times. No significant difference in performance between simple and complex query types was observed. Since there were no differences in query durations, and since HBase is linearly scalable, it was expected that query durations would decrease with an increase in the number of nodes and be within a few milliseconds as nodes approached 100, even

at ten billion rows. The study, therefore, established an innovative BDA platform that met the performance requirements for querying patient data in a NoSQL database while significantly reducing the time required to retrieve patient data.

At first it was difficult to ascertain why HBase took so long to show as completed. Meeting the goal of establishing the entire three billion or 30TB of data, in less than a week, initially seemed almost impossible, even with the addition of extra cores and nodes. The tasks fitting a similarity of a MapReduce solution, as well as large-scale platforms with the MapReduce paradigm were found when Hadoop employed a MapReduce execution engine (Dean & Ghemawat, 2004; Dean & Ghemawat, 2010) to implement its fault-tolerant distributed computing system over the large data sets stored in the cluster's distributed file system. It was difficult to understand not only why the ingestion of files was slow but also why a few of the servers kept dying. This ingestion process comprised the bulk of the framework. MapReduce, at a framework level with HBase, had drastically slower indexing performance; what would have taken a week or two to ingest (three billion rows) took over a month and involved daily ingestion followed by compaction before the next ingestion. It is also important to note that VIHA's total data warehouse comprises ~500 million encounters at 4–14 TB (depending on whether non-structured data is included); this study's data set was almost three times that size and included almost six times the number of encounters.

The MapReduce programming framework has been used to classify biometric measurements using the Hadoop platform for face matching, iris recognition, and fingerprint recognition (Kohlwey, Sussman, Trost, & Maurer, 2011). For bioinformatics, it has been used for genome and protein Big Data analysis using the MapReduce framework (Chen, Hung, Tsai, & Lin, 2014). The large datasets stemming from genomic data are particularly amenable to analysis by distributed systems, unlike data from hospital systems. A genome-sequence–comparison algorithm devised by Zhang, Sun, Waterman and Chen (2004) has been implemented on top of Hadoop while relying on HBase (Nguyen et al., 2011) for data management and MapReduce computation. The Big Data technologies used, including HBase, Hadoop, and MapReduce, indicate the range of platforms that can be applied to genetics, microbiology, and bioinformatics. However, the construct used in this study was very different from the simulation of data of the hospital system over the platform than matching genes to patient-specific treatments because the data could not be represented by libraries and matched in real time. Nevertheless, the study showed what is required to maintain a productive system and provided a technical sense of how jobs run on MapReduce and how interfaces are structured for queries at extremely large volumes.

5.2 Data Modeling of Patient Data of Hospital System

Big Data in healthcare can cover tens of millions of patients and present unprecedented opportunities. Analyzing patient-level databases can yield population-level inferences or "results" (such as the strength of association between medical treatments and outcomes), often with thousands of outcomes. Although data from such sources as hospital EHR systems are generally of much lower quality than data carefully collected by researchers investigating specific questions, the sheer volume of data may compensate for its qualitative deficiencies, provided that a significant pattern can be found amid the noise (Hansen et al., 2014; O'Sullivan, Thompson, & Clifford, 2014). In addition, there is a trend toward higher quality Big Data collection (such as the data produced in genomic analysis and structured data that can be generated from standard-compliant EHR systems) (Freire et al., 2016; Frey, Lenert, & Lopez-Campos, 2014). On the other hand, Mayer-Schönberger and Cukie (2013) showed that as the percentage of the population being sampled approaches 100%, messy data can have greater predictive power than highly cleaned and carefully collected data that might represent only a small sample of the target population. The present study did not analyze the structure of the metadata. Ultimately, it was designed not only to replicate data but to simulate the entire volume of production and archived data at Vancouver Island Health Authority (VIHA) and possible the province such that real patient data from hospitals at VIHA will be approved to utilize over the platform. Therefore, the messiness of the data and its influence on the simulation was not tested, although this could potentially affect accuracy and performance when querying real data.

The data used in this study consisted of diagnosis codes, personal health numbers, medical record numbers, dates of birth, and location mnemonics (to mention only a few of the 90 columns), as these codes are standardized for hospital systems and, compared to genetic data, easier to replicate in metadata in large volumes. The use of groups of events allows the definition of a phenotype to go beyond diagnosis as coded using the International Classification of Disease, version 9, codes (ICD-9) and potentially allows assessment of the accuracy of assigned codes (Freire et al. 2016; Hripcsak & Albers, 2013). In healthcare, the complexity of Big Data storage and querying increase with unstructured sets of data and/or images. Images take up lots of storage capacity and are difficult to process and next to impossible to query in large volumes. The growth in the volume of medical images produced on a daily basis in modern hospitals has forced a move away from traditional medical image analysis and indexing approaches towards scalable solutions (Wang et al., 2011). MapReduce has been used to speed up and make possible three large-scale medical image processing use-cases: (1) parameter optimization for lung texture classification using support vector machines (SVM), (2) content-based medical image indexing/retrieval, and (3) dimensional directional wavelet analysis for solid texture classification (Markonis, Schaer, Eggel, Müller, & Depeursinge, 2012). In their study, as in our study, a default cluster of heterogeneous computing nodes was set up using the Hadoop platform, allowing for a maximum of 42 concurrent Map tasks. The present study did not test the amount and efficiency of concurrent Map tasks of MapReduce to process the data to HBase ingestions; this is something to be investigated further as using real hospital data that is highly unstructured and rich in images might require this enhancement. Moreover, this study ran up against limitations in the ability of

the Reducer component of MapReduce more than Map tasks to form the bulkloads of HBase and its NoSQL schema, and, therefore, the Reducer improvements should be investigated before Map tasks.

The ADT data is very difficult to emulate because it is from Cerner Millennium System, which uses a kernel to create alias pools for 1000 different tables in the database. Patients can have multiple patient encounters per visit, a factor not simulated in the database in this study. Thus, encounters were simplified in this study's database with one row representing an encounter with random values. Simply creating one flat file cannot emulate the metadata relationships and does not guarantee the data are treated uniquely for each encounter row when the encounters can change over time or several are linked to the same patient. However, if the data is extracted from the automated hospital system and it is confirmed that the columns are correct with unique rows, it should be possible to combine it with DAD data with similar unique keys and qualifiers. Again, DAD and its diagnoses and interventions are not highly correlated or linked to the ADT system, which depends on several factors, such as clinical workflow, the most responsible provider, the type of provider, details of reporting, manual classifications, and how and when health outcomes were determined. Currently, there are very few columns linking ADT to diagnosis or it's DAD, and these few are primarily admission source and encounter types in the hospital system, including admission time and date and discharge time and date.

The complex nature of HBase means that it is difficult to test the robustness of the data in emulations based on real data. This complexity somewhat rejects the hypothesis that noSQL database accurately simulates patient data. Nevertheless, several steps are stilled required by hospitals to prepare the DAD database alone for statistical rendering before sending it to CIHI. Moreover, the actual columns used in this study are the ones used by VIHA to derive the information accurately in a relational database, which ensures the data is in alias pools and not duplicated for any of the encounters. The DAD data also makes calculations (which in the reporting workflow adds columns), which is what the reporting and data warehouse teams also do to form their databases. Adding columns to a NoSQL database is much easier than adding columns to a SQL relational database, and von der Weth and Datta (2012) showed good performance of multi-term keyword searches. Therefore, it is an advantage to have a large database with row keys and column families already set up; this is supported by Xu et al. (2016), as their middleware ZQL could easily convert relational to non-relational data. However, the indexing of HBase proved to decrease the time to ingest the data accurately (so that queries produced accurate information). If the indexing was not reiterated, with the addition of new columns or rows, then the data cannot be queried at all. Spark and Drill performed well with the larger volumes, thus offering an alternative to HBase (without the indexing); however, the data cannot be fully represented without indexing if real patient data is to be used. Not all columns are known, but mimicking an index with family columns does enable simulation of the data model of the hospital. This is a significant drawback, and more data modeling of the relational to the non-relational database is required.

It was more complicated to validate the simulated data in Spark and Drill with real data. Scott (2015) indicated that the battlefield for the best big data software solutions is

between Spark and Drill and that Drill can emulate complex data much more efficiently than Spark because Spark requires elaborate Java, Python and Scala coding to do so. Nonetheless, both Spark and Drill were significantly faster than HBase in ingesting files directly into Hadoop via *Drillbits* (Drill) with ZooKeeper and MapReduce, and RRD transformations with MapReduce (Spark). In fact, the ingestion and queries for both Spark and Drill could be run in sequence instead of having to run compaction as well. However, it is difficult to compare the data emulation of HBase to that of Spark and Drill. Neither Spark nor Drill indexed the files, nor did they require the Reducer to complete its task before queries could be performed. The tools used totally different processes across the nodes, and without indexing there is a lack of encrypted data (which patient data requires); those processes did, in the end, produce the same query, but that is because the platform was set to ingest the already-indexed files into Spark and Drill. Absence of indexing increases the risk of inaccuracies (even though the framework was more fault-tolerant when running Spark and Drill). Therefore, the big data tools and inherent technologies highly influence the health informatics of the data established and resulting data from queries.

Other research reviews (e.g., Kuo et al., 2015; Nelson & Staggers, 2014; Li et al., 2016; Wang et al., 2014) have stressed the importance of patient data modeling with Big Data platforms in healthcare, indicating that a lack of BDA ecosystems is one of the reasons why healthcare is behind other sectors in utilizing current technologies to harness Big Data. Nelson and Staggers (2014) noted that nursing informatics and data from nurse progress notes are underutilized in hospital systems. Wang et al. (2014) also compare bioinformatics with healthcare and Big Data applications. Bioinformatics can match extremely large libraries of genetic data to libraries of medications or treatments; however, such matching cannot be performed at the scale of large hospital systems, and patient-centric frameworks and current traditional practices of storing relational data make it difficult to replicate for other database types, especially Big Data. Chawla and Davis (2013) and Kuo et al. (2011) argue that even structured data lack interoperability among hospital systems, so that no solutions could possibly link all data. At VIHA, for example, it is difficult to link the DAD and ADT data on encounters, because the DAD data on diagnosis and intervention are not stored together or integrated or has relational dependencies in an all in one data warehouse, while the ADT automatically links the data to encounters (Nelson & Staggers, 2014; Kuo et al., 2011). Therefore, more validation is required to match corresponding medical services in ADT to patient diagnosis from the DAD in that admission time and source.

Essentially this study is proposing a row-column key-value (KV) model to the data distributed over a customized BDA platform for healthcare application. Wang, Goh, Wong, and Montana (2013) support this study's claim in their statement that non-relational data models, such as the KV model implemented in NoSQL databases. Wang et al. (2014) further stated that NoSQL provided high performance solutions for healthcare, being better suited for high-dimensional data storage and querying, optimized for database scalability and performance. A KV pair data model supports faster queries of large-scale microarray data and is implemented using HBase (an implementation of Google's BigTable storage system). The new KV data model

implemented on HBase exhibited an average 5.24-fold increase in high-dimensional biological data query performance compared to the relational model implemented on MySQL Cluster and an average 6.47-fold increase on query performance on MongoDB (Sakr & Elgammal, 2016). Freire et al. (2016) showed highest performance of CouchDB (similar to MongoDB and document store model) but required much more disk space and longer indexing time compared to other KV stores. The performance evaluation found that the new KV data model, in particular its implementation in HBase, outperforms the relational model currently implemented, and, therefore, supports this studies NoSQL technology for large-scale data management over operational BDA platform of data from hospital systems.

It is possible to mimic a relational database with an SQL-like structure using NoSQL (e.g., Xu et al., 2016). However, when this was done in the present study, the primary key strength of the patient encounter was somewhat altered when combined with other parameters that are also primary keys. In addition, the key values must be unique; therefore, when replicating the ingestion, queries had to be modified; otherwise no result would be obtained. Thus, it was proven that dependencies in the inherent system could be expressed in SQL-like queries over Apache Phoenix to generate results from the patient data. Chawla and Davis (2013) emphasize the importance of the patient-centered framework for the Big Data platform. Since many Big Data platforms are linked to HDFS with non-relational databases, assuring the accuracy of patient data is of the utmost importance. This was achieved in simulating queries on the NoSQL database but not fully validated over the emulated database with non-replicated data of more detailed encounter-based patient queries.

5.3 Hadoop/MapReduce Framework on Platform

The framework Hadoop/MapReduce/HBase with Phoenix, Spark, and Drill allowed for further investigation of the performances on the simulated data. In this study, a platform using this framework was successfully built and tested with emulated patient data. Achieving a productive system necessitated a number of configuration changes that drastically influenced the replication of the database via datasets. Replication was improved by pre-splitting the table via “salt buckets” in the Apache Phoenix table creation. Though salt buckets in the formula and module load improved the replication, it was still limited by the amount of disk space on any particular node; by CPU memory allocation, such as RAM consumption; and by the duration of the MapReduce component in Hadoop. This finding accepts the hypothesis of high performance at maximum core CPU and corresponded to the results of Ruay-Shiung et al. (2014) and Yang, Liu, Hsu, and Chu (2013), who found that table space on the database node, is a major barrier for Hadoop since Hadoop cannot momentarily stop ingestion on one node to switch to the other node.

Mohammed et al. (2014) stated that Hadoop/MapReduce platforms are emerging for use with large-volume clinical datasets derived from laboratory information system (LIS) data, test utilization data, electronic medical records (EMR), biomedical data, biometrics data (Jonas, Solangasenathirajan, & Hett, 2014), gene expression data, and other sources. However, very few Big Data technologies have been performance tested for use with

healthcare, health informatics, and clinical databases and the results compared to existing reporting systems. Additionally, there are very few usability studies of BI tools, data mining, and Hadoop's ecosystem in healthcare. There is increasing complexity and advanced software in Hadoop's ecosystem, with growing application to bioinformatics but not to clinical databases. This increase towards bioinformatics compared to health informatics might be due to the fact that the workflow in clinical settings is complicated with clinical standards depicting the daily work activities and metadata standards representing the data stored regardless of free text or other types outside the standard. Hence, usability is difficult to verify without another full study and cannot be validated or even tested using Big Data technologies over the established platform at this time. Furthermore, massive datasets are extremely difficult to analyze and query using traditional mechanisms, especially when the queries themselves are quite complicated. In effect, a MapReduce algorithm and its components of the query can be processed simultaneously—or reduced—to rapidly return results. Without Hadoop and MapReduce, this study would not have achieved the iterations required for large volumes of complicated structured data.

Hadoop divides huge chunks of data into smaller chunks and distributes the process across multiple machines (six nodes over the platform in this case), thus producing results quickly. To avoid any data loss, the platform makes sure it is configured (but the operational outcome depends on several factors like network connectivity and actual bandwidth acquired at that point in time) that each chunk of data runs in parallel on different machines; with the use of the "salt buckets" command in Phoenix-HBase, a balanced distribution is obtained. In order to prevent data loss and improve network performance, the Hadoop administrator manually defines the rack number of each slave DataNode in the cluster. Moreover, Rallapalli, Gondkar, and Ketavarapu (2016) reviewed Hadoop clusters as built rack servers connected to the-top-of rack switch. Uplinks of a rack switch are connected to another set of switches of equal bandwidth or Ethernet and are linked to InfiniBand. This forms a cluster in a network. In this case, the researchers loaded the EHR data to the cluster and executed a query, which also Wassan (2014) accomplished in a procedure similar to this study's proof-of-concept implementation. The steps with their platform and one established in this study were almost identical, in that the workflow of the cluster consisted of HDFS writing the loaded data (as HFiles) into the cluster (or Hermes nodes at WestGrid) before data was placed and analyzed with MapReduce algorithms. Essentially, HDFS reads the results from the cluster. In their study, from large sets of EHR data, they showed that their system could find how many patients were diagnosed with heart disease and analyze and process this data very quickly using Hadoop. The present study did not investigate the hardware or existing architecture at WestGrid. However, it did review and maintain a balanced ingestion on the dedicated servers.

Ruay-Shiung et al. (2014) showed that Hadoop could be built to re-configure duplication of the data generated such that testing the accuracy of the data could be done efficiently and across different distribution processes. Jurney (2013) also suggests using actions, predictions, reports, charts, and records as a model for Hadoop batch processes, and query tool configurations to take into consideration at a technical level before testing

queries. An analysis of the data and of how Hadoop distributes the files with HBase bulkloads proved that queries generated patient data according to the established specification replicated to create large volume, but only after adding the primary keys could the data be displayed. Therefore, it was proven that the emulated patient data achieved integration, but the platform could not be tested for data quality. Moreover, data quality in BDA application with Hadoop is difficult to test in an exact replication of its relational database origins (Dhyani & Barthwal, 2014). The present study did not run machine- learning algorithms on the replicated data as the algorithms would not have revealed any real patterns in the data. Since the data is replicated, only a random pattern can be found, and this holds true for using visualizations to explore the simulated data.

The promise of BDA in bioinformatics and healthcare has previously been described (Baro et al., 2015; Raghupathi & Raghupathi, 2014; Twist et al., 2015). Dai et al. (2012) and Taylor (2010) provided an overview and discussion of the Hadoop platform, MapReduce framework, and its current applications for the field of bioinformatics (see also Mohammed et al., 2014; Sakr & Elgammal, 2016). The review by Baro et al. (2015) enlarges the scope to the application of the MapReduce framework and its open-source implementation using Hadoop to a wide range of clinical Big Data. Their study, however, only enlarges the scope of Big Data technologies over platforms applied to biometrics and matching bioinformatics native libraries of DNA and medical treatments and not to health informatics at hospitals. Although MapReduce frameworks have been applied to DNA sequencing data by McKenna et al. (2010), matching libraries in real-time instead of finding trends is the main application of MapReduce in bioinformatics. The two are very different in the sense that data at hospitals is very heterogeneous and located in many relational tables while the bioinformatics data is streamed through an already-established pipeline. Accessing the stream from a platform establishes the known bioinformatics data elements or libraries; however, in the case of health informatics, little is known about the data in the database, about who is entering it, or about what automated system parts are storing it.

Many state that parallelization is the key to the success in running Hadoop-MapReduce at high levels of performance (Karim & Jeong, 2011; Lith & Mattson 2010; Maier, 2013; Yu, et al., 2016). Our study did change some of the jobs to effectively run in parallel, but the architecture and nodes used were serial in their initial access, so that running Hadoop from the head node to worker nodes required manually accessing the modules to run jobs each time. Additionally, since there was little difference between the SQL complexity types over Apache Phoenix (although both Spark and Drill were faster), and since HBase is linearly scalable over the indexed (desired) rows, the query time durations are expected to decrease with the number of nodes; they should fall to within a few milliseconds at around 100 nodes even at a width of 10 billion rows. However, testing parallel performance over the big technologies was not fully carried out in this study. The study did test some of the parallel processes running on default configurations without using InfiniBand, ZooKeeper, or RegionServers, which drastically slowed the ingestion of the files and overall performance if not completely processing in parallel. One study showed notable speed-ups in processing after a parallel version of an advanced algorithm (Díaz-

Uriarte & De Andres, 2006) for regression and genetic similarity learning tasks had been developed (Wang et al., 2013) for large-scale population genetic association studies involving multivariate traits. Another study proposed a solution to sequence comparison that thoroughly decomposed it into multiple rounds of Map and Reduce operations; however, this could only be achieved with parallel computing and not serial or sequential programming methods (Almeida, Grüneberg, Maass, & Vinga, 2012) under multiple sequencing alignments (Sadasivam & Baktavatchalam, 2010). Taking an innovative approach, Nguyen, Wynden, and Sun (2011) showed that clinical signals could be stored and processed using a platform with Apache HBase distributed column store and the MapReduce programming framework with an integrated Web-based data visualization layer, and ultimately this form of visualization is important to establish in nursing informatics and education (Skiba, 2014). Therefore, since this study's platform did utilize HBase distributed column stores, there is room to apply more advanced algorithms to the Hadoop/MapReduce framework.

The present study did not investigate the architectural design of the data warehouse it created, since WestGrid provided the architecture. Instead, using Big Data technologies in frequent application to generate large volumes of data was the ultimate goal. The study determined that Hadoop/MapReduce-based clustering worked well with the hospital data. Studies of clustering have been conducted by, among others, Ngazimbi (2009) and Heafield (2008) at Google (Hadoop design and k-means clustering). However, the literature suggests that there are some drawbacks to HDFS use (Maier, 2013; Taylor, 2010). HDFS does not handle continuous updates as well as a traditional relational database management system could. Moreover, Hadoop had been shown to be an inadequate platform for scalable analysis of streaming data (Sakr, Liu, & Fayoumi, 2013). Twitter released the Storm system, which fills this gap by providing a distributed and fault-tolerant platform for implementing continuous and real-time processing applications of streamed data. Storm cannot be used to combine hospital databases or to execute queries. However, it was used at a hospital in Kansas City to link DNA with possible treatments, where Apache Storm took 26 hours to suggest possible treatments for children with unknown conditions (based on symptoms). Additionally, Sakr and Elgammal (2016) stated that HDFS cannot be directly mounted onto an existing operating system. Therefore, if hospitals and health authorities want to use Big Data technologies, they need to apply the technologies as a build-to-implement rather than a bolt-on middleware to the system. In short, getting data into and out of the HDFS file system can be awkward. Chang et al. (2014) have shown that Big Data can be generated and updated tremendously fast by users through a variety of devices anytime and anywhere. Coping with sets of multiform data in real time is very challenging. Similarly, the present study found that using keystores doubled the size of the files when ingested via Hadoop to HBase. Correspondingly, Chang, Liao, Fan and Wu (2014) show that using the deduplication technique with HDFS can improve the efficiency with which storage space is used and reduce duplication inherent with Hadoop ecosystems and subsequent keystores.

5.4 NoSQL Database Using HBase

HDFS with HBase was designed to meet Big Data challenges by building a distributed data center, using data duplicates to increase reliability. However, data duplicates take up a lot of extra storage space, so this approach requires ample funding for infrastructure. Traditionally, structural data have been normalized in advance, stored in databases, and then manipulated as the principal resource to support enterprise IT systems. Other data, which are unstructured or semi-structured and generally larger in volume than structural data, are hard to process and are cast aside or trashed. However, as new cloud-computing technologies (like Hadoop and NoSQL) have emerged (Chen et al., 2015), this other data is now considered the most valuable resource, and enterprises have begun making strides in developing a new market for it. Issues like gathering, storing, modeling, analyzing, and manipulating Big Data have turned out to be important in cloud-computing research and applications. When Big Data is mentioned now, what comes to mind are no longer the hegemonic past systems of the database market Oracle or software giant Microsoft (Jorgensen et al., 2014), but the Apache Foundation's open-source Hadoop parallel computing and storage architecture and the HBase NoSQL distributed database.

The present study showed that performing maintenance and operational activities over the platform were essential for ensuring reliability. Data was replicated across multiple nodes, but there were unbalanced ingestions that required removing files and starting again. Some studies have shown that Hadoop can detect task failure and restart programs on healthy nodes, but if the RegionServers for HBase failed, this process had to be started manually (Chang, Chen, Ho, & Chen, 2012; Chung, Lin, Chen, Jiang, & Chung, 2014; Dutta & Demme, 2011; Lith & Mattson, 2010; Maier, 2013). Our study showed that compaction improved the number of successful runs of ingestion; however, it did not prevent failure of the nodes, a finding that is supported by other studies (e.g., Dean & Ghemawat, 2010; Greeshma & Pradeepini, 2016; Mohammed et al., 2014; Nabavinejad et al., 2016; Taylor, 2010). If a node failed, the partly ingested file had to be cleaned up, re-run, and re-indexed. The platform, therefore, did show a single point of failure that remained at the NameNode for the HDFS file system. However, despite failures when first running the ingestions, the actual queries did not show any major fluctuations in performance.

Data locality is important, especially for the simulation of patient data. Hadoop tries to automatically co-locate data with the computing node. That is, on the platform with the generation of HFiles to use MapReduce to ingest the files to form the database, Hadoop schedules Map tasks close to the data on which they will work, with "close" meaning in the same node or, at least, the same rack. The investigation found this to be a principal factor in Hadoop's performance, especially with HBase bulkloads to simulate the set metadata of the hospital to form the indexed database. Performance checks of Hadoop were conducted in April 2008: A Hadoop program, running on a 910-node cluster, broke a world record, sorting a terabyte of data in less than 3.5 minutes (Gray, 2008). This study did not achieve performance at all close to that; at ca. 10-12 hours for 30TB to sort the data using only six nodes the platform did achieve high performance to reach three billion (indexed) records simulated. Speed improvements continued as the Hadoop

configurations were finally set with 3–6 minutes for 24GB into Drill. With HBase, however, it was very difficult to get performance under 300 minutes for each 50 million (which was only 258MB). There were major fluctuations in performance between ingestions, from 3–4 hours on some occasions to 10–12 hours on others. These fluctuations might be due to the construct of the framework and the fact that the indexing of HBase created the platform's major bottleneck and limitation on performance via MapReduce.

Although the construction of the framework of a BDA platform for health applications is still in an early stage, the present study showed that existing NoSQL solutions in the Big Data realm of its application should be considered for use in addressing the healthcare challenges of analyzing large volumes of data. Emulation of combined ADT and DAD data to form a database was accomplished. It was proven that dependencies in the inherent system could be expressed in "family" primary keys via Apache Phoenix to generate the database and query the data accurately. Therefore, key-value (KV) storage via HBase did not hinder performance when adding an ANSI-SQL:2003 or SQL-like layer on top of the framework. However, the emulation of the ADT system of alias pools for encounters in a patient-centric system did not include all primary keys and superimposed many primary keys as column families across the emulated patient data as a distributed table. Emulating the ADT system of alias pools for encounters in a patient-centric system was difficult; not all primary keys were included because they some did not coincide with the data profiles of the DAD.

Further regression testing of the HBase database (over an interface) to represent unique encounters within the 90 columns will require investigation. Even though the platform was able to combine the ADT and DAD databases (with similar metadata to VIHA's), this study did not prove that any query on a patient encounter will span both ADT and DAD data in accordance with VIHA's data warehouse policy and constructs. More investigation is needed to establish a data model for the NoSQL database in order to compare it with VIHA's relational database. VIHA and most hospitals do not query or report on both ADT and DAD in their data warehouses simultaneously. Such queries as the impact of frequent bed movements on patient outcomes over the entire hospital system could be analyzed at a larger hospital level of patient care using the BDA platform. Accessing the real data could allow for the combination of ADT and DAD analysis, but only in a specific Big Data context or model.

In this study, it converted experimental data from Excel document files to HBase without time limits, and with no high fault-tolerance considerations. The purpose was to find the most efficient combination of Hadoop, its distributed file system, and a configuration of available query tools. This study's initial data generator process was supported by Yang et al. (2013) proposed converting the format of Excel data and storing them in a database based on HBase on Hadoop frame. Furthermore, studies have shown that HBase is a non-relational database with a structure similar to Excel but with higher implementing feasibility (Nishimura et al., 2012; Sun, 2013; Vashishtha & Stroulia, 2011; Zhang, 2010). For medical data usage, which requires fast, accurate, and efficient querying of the required information for people in various areas, it provided a visual graphical interface,

instead of using query languages, to speed up data queries. Unlike traditional commercial relational databases, HBase is scalable, high-performance, and low-cost, but it is not complete or user-friendly tested, which this study's operational investigation did identify.

Unlike relational database systems, HBase itself does not support SQL; in fact, HBase is not a relational data store at all. HBase applications like Apache Phoenix can provide an SQL-like layer over HBase and are written in Java, much like typical MapReduce applications. The platform used in our study had ran into the problem of RegionServer hitting the InfiniBand correctly and fully, as well as the settings to run compaction after each ingestion did not always compact the files correctly, which caused the entire operational iteration of ingestion to fail. HBase settings also had to be purged or cleaned after each of the ingestions due to unknown tracers or remnants of transactions that then later caused failure. Every HBase system comprises a set of tables. Each table contains rows and columns, much like a traditional database. Each table must have an element defined as a Primary Key, and all access attempts to HBase tables need to use this Primary Key; even Apache Spark SQL can be linked to database conversions with HBase (Xu et al., 2016), and other transformation methods can be applied to healthcare data towards an intact NoSQL database (Yang, Liu, Hsu, & Chu, 2013). Additionally, in this case of construct, HBase column represents an attribute of an object; for example, if the table had logs from servers, where each row might be a log record, a typical column would show the timestamp of the time when the log record was written, or perhaps name of the server on which the record was originated. Thus, HBase allowed for many attributes to be grouped into so-called column families and stores the elements of the column family together. This is different from a row-oriented relational database, where all the columns of a given row are stored together.

Though three billion rows of data were achieved, the study's operational activities were forced to customize the open-source construct for both HBase and Hadoop because the ingestions were fast on the Map portion but not the Reduce portion (which takes into consideration the indexing of HBase and key stores). In fact, it was found that at each of the iterations, the ingestion of another 500 million files into the replication was slower on account of the re-indexing of all the rows in the key stores. Indexing was essential because it formed the patient encounter, which was the main primary key, and without complete indexing the data could not be queried. HBase indexing creates indexes at each of the iterations across the flat file on all the nodes and is not continuous at each replication. Moreover, the study utilized a columnar (row-column-oriented) type of NoSQL database over a platform. In this application, data from a given column was stored together, in contrast to a row-oriented database (e.g., a relational database system), which keeps information about each row together. In column-oriented databases, the process of adding new columns is quite flexible and can be performed on the fly on a row-by-row basis. In principle, columnar NoSQL systems represent occupy the middle ground between relational and KV stores. Apache HBase is currently the most popular open-source system in this category (Sakr & Elgammal, 2016); it was selected because of its popularity in the row-and-column category and used more widely than other databases, so that blogs and code could be exchanged more easily with users. Its selection to simulate hospital data is still recommended even though there were some

operational delays to form the datasets over large volumes. The HBase database did provide insight into three main categories to further investigate: how data can be continually updated over the distributed nodes; how indexing, key stores and primary keys can be applied; and how columns can be added in the row-column matrix. All of these three insights are currently applied at VIHA's data warehouse and, therefore, findings can be cross-referenced at large scale over the database of real data.

With HBase, the table schema is dynamic because it can be modified on the fly and can incorporate other forms of data for different schemata (Scott, 2015). However, in order for the platform to work and function, HBase's schema must be predefined (this was done via SQL-like Phoenix in this study) and the column families specified before it is operational. However, the schema is very flexible, in that new columns can be added to families at any time; it is therefore able to adapt to changing application requirements (Nishimura et al., 2012; Nguyen et al., 2011). As Sun (2013) states, "Just as HDFS has a NameNode and slave nodes, and MapReduce has JobTracker and TaskTracker slaves, in HBase a master node (HMaster) manages the cluster and RegionServers store portions of the tables and perform the work on the data." HMaster is the implementation of the Master Server, which is responsible for monitoring all RegionServer instances in the cluster. In this study's distributed cluster, the Master started on the NameNode, while HRegionServer started the RegionServer(s). In a distributed cluster, a RegionServer runs on a DataNode (Sun, 2013). In HBase, through the ZooKeeper, other machines are selected within the cluster as HMaster (unlike the HDFS architecture, in which NameNode has a single-point-of-availability problem) (Sun, 2013). HBase clusters can be expanded by adding RegionServers hosted on commodity class servers. For example, when a cluster expands from 10 to 20 RegionServers, it doubles both in terms of storage and processing capacity. Therefore, more storage space will be required when the number of nodes increases and the investigation uses real patient data. Sun (2013) lists the following notable features of HBase: strongly consistent reads/writes; "*Automatic sharding*" (in that HBase tables distributed on the cluster via regions can be automatically split and re-distributed as data grows); automatic RegionServer failover; block cache and "*blooming*" filters for high-volume query optimization; and built-in web-applications for operational insight along with JMX (i.e., Java memory) metrics. The present study did not investigate the robustness of HBase and cannot verify Sun's (2013) findings. However, it did confirm that HBase supports HDFS out of the box as its distributed file system; that HBase supports vastly parallelized processing via MapReduce; and that HBase provides a user-friendly Java API for programmatic access. Further investigation of APIs might also reveal better interfaces than Apache Phoenix for querying data.

5.5 Security and Privacy over Platform

In Canada's healthcare system, population health data policy is to publicly disclose data when it is externally stored outside an authority's network boundaries (Dufresne et al., 2014; Pencarrick Hertzman et al., 2013). The proposed BDA platform utilized existing architecture at WestGrid at UVic, external to VIHA. WestGrid maintains a secure environment by restricting access to accounts, and the Hadoop/HBase ingestion processes to generate and store data cannot be accessed by anyone other than the currently authorized user. Thus, the BDA platform is highly secure. To replicate data from source to HBase to form one billion patient encounters required one week; the data needs to be stored before running the queries. If data is stored with longer than a few hours without removal, public disclosure is required by laws concerned with sensitive personal data (Moselle, 2015). Furthermore, Dufresne et al. (2014) point out that Canadians have a nationalistic view of their healthcare system, and public disclosure of data in a publicly provided healthcare system makes sense to many citizens, including providers and health professionals. Thus, if the BDA platform can produce accurate query results within seconds while the data is still housed at the health authority, or if the data is encrypted and highly secure, then public disclosure might be deemed unnecessary. Conversely, the introduction of Big Data technologies could lead to a change in disclosure policy.

Wang et al. (2015) emphasize that combining patient data in hospital systems can be achieved using Big Data, though few studies have done so. The present study showed that HBase provided an index and key store for patient-related data necessary for querying the files after running MapReduce. This index is very important to the performance of the queries, as it allowed all queries to retrieve any and all information requested while encrypting the data during any batch job, including its replication via the bulkloads. The data generated by HBase cannot be viewed directly, and the sequence of processing from HDFS to HBase to Phoenix must always be performed to query and view the data. It was found that Hadoop HDFS had inherent processes that were auto-correlated with HBase in the configuration and process sequence (cf. Dutta & Demme, 2011; Spivey & Echeverria, 2015; White, 2015), which allowed for high performance accurate queries under severely restricted data access.

Sending data over to different nodes takes time, potentially several hours. This duration could prove to be a security weakness (e.g., Alvaro, Cardenas, Manadhata, & Rajan, 2013) or increase the vulnerability of the data to unauthorized access while it is in transit. The WestGrid architecture is hierarchical with one master node and several subordinate nodes; however, each node can be accessed, depending on the LDAP environment. Access is three-layered, with the WestGrid administrator sponsoring the person who sponsors others. Levels of access vary; the administrator has more capacity to modify and change the configuration, while the others' access is read-only.

Using HBase to ingest files, data could only be queried in specific sequence with a login account. Spark and Drill, on the other hand, did not require indexing and were relatively quickly placed into Hadoop to run queries. Furthermore, to be fault-tolerant in the unshared architecture, tasks must have no dependence on each other, with the exception of mappers feeding into reducers via MapReduce under Hadoop control (which was not

entirely confirmed). Therefore, even though there are security risks with open-source software, the number of versions and advanced customization decrease the risk of malware affecting configurations on the system. Hadoop was primarily used to allow distributed storage and audit data. *Sqoop* is a software tool that can transfer data between Hadoop HDFS and structured databases, such as MySQL and HBase (Sitto & Presser, 2015), and used in this study's framework. Chen and Fu (2015) conducted Hadoop storage experiments in which five computer nodes were used to build a cloud-storage system. One of these was the NameNode; the other four are DataNodes. The import instruction of Sqoop is used to periodically import data tables from the MySQL database to Hadoop HDFS. The data tables included a Sunspot security entity table and a Sunspot measure table. Only a security entity table was exported to Hadoop HDFS. To ensure the security of the audit, only security administrators were granted access to the audit data stored in Hadoop HDFS. If a security administrator suspected that the security entity table had been tampered with, the data on the Hadoop HDFS could be checked to detect any tampering.

A table in HBase is made up of regions of rows and columns. Each region is defined by a start key and an end key, as well as by primary keys, which may exist on different nodes; each key is made up of several HDFS files and blocks, which in turn are replicated by Hadoop (Sun, 2013). Columns can be added to tables on the fly, with only the parent column families being fixed in a schema. The data model was based on supporting faster queries over large-scale schemata using HBase. Dutta and Demme (2011, p. 56) noted for a "typical use case when multiple identical Patient IDs are retrieved, the Store Files storing these patients' data will be loaded into BigTable caches." Each cell is tagged by column family and column name, so programs can always identify what type of data item a given cell contains (Nguyen et al., 2011). Secondary indices are possible through additional index tables, but this study did not use them. Noteworthy advantages of the platform included the ability to scale to petabyte-size datasets and the ease of integration of disparate data sources into a small number of HBase tables for building a data workspace, with different columns possibly defined (on the fly) for different rows in the same table. However, only retrieving the first record this study showed slowness in performance, which was also shown by Dutta and Demme (2011) in that it causes a significant delay when loading data from disk to memory with other records in the same Store File fetched from memory caches to speed up the query. This design takes full advantage of BigTable caches via HBase keys, although the number of caches available increases security risks; using indexed rows in HBase significantly reduces these risks. HBase provides a platform with many types of keys and qualifiers in its column-row key-store, which can heavily encrypt the data and increase security and privacy, especially when the data is distributed.

Karim, Ahmed, Jeong, and Choi (2013) removed the drawbacks of a relational database management system (RDBMS) and main-memory-processor-based computing to facilitate the existing MapReduce framework (see also Agrawal & Srikant, 1994; Ekane & Fox, 2008; Elsayed, Lin, & Oard, 2009; Dean & Ghemawa, 2004; Dutta & Demme, 2011; Karim, Jeong, & Choi, 2012; Karim & Jeong, 2011). Another study using a similar BDA framework removed null transactions and infrequent items from the segmented

dataset, sorted the items in ascending order, and then applied parallel Frequent Pattern-growth (Zhou, Jong, & Feng, 2010) to generate the complete set of frequent item sets with the improved *ComMapReduce* framework (Ding & Huang, 2012) on Hadoop. The platform was able to generate the complete data set of an in-hospital patient encounter with subsequent diagnoses and treatments, ruled by means of maximal frequent item sets, revealing interesting patterns. It remains to be seen whether the NoSQL database can produce constraints that mimic those of relational databases. Also, even though query results were benchmarked in our study, benchmarking real data will be much more data-intensive and rely heavily on Big Data technologies being configured correctly and according to the desired data model. There have been no studies on the benchmarking of hospital data under privacy/security using Big Data technologies. Other studies achieved accurate linkages between large datasets (genetic DNA libraries and treatment libraries) as a match scenario rather than recording multiple-layered patient encounters in a single database (Karim et al., 2014; Miller et al., 2014), but privacy/security was not validated in their studies.

5.6 Future Work

Big Data solutions present an evolution of clinical big data analysis necessitated by the emergence of ultra-large-scale datasets. The volume size requirement of Big Data presents a major challenge for future work to find any online datasets, fake, de-identified or real, at ultra-large scale. There are extra hassles to requesting large volumes of patient data because many stakeholders need to be involved, like physicians, nurses, executives, data warehouse, database administrators, clinical reporters, and application platform with bolt-on solutions (e.g., SQL parsers). Not just one stakeholder needs to be involved because of the work and resources required to initially move the data. Hadoop can be configured to move data from different locations and distribute to others; however, the movement of data needs to maintain data integrity for accuracy and enforce privacy/security continually. I initially had meeting after meeting to not only understand the data but get approval to use a small (at first) to then get approval of larger datasets. All VIHA staff and experts of the data at the hospital met more than once but the main barrier was to get permission to allocate their time (i.e. three months) for project work in the initial data migration tests and deployment at large volumes.

The actual advantages of big data technologies were not seen clearly by most groups because none of them used these tools at all. However, clinical reporting team did express keen interest in the end product and had many ideas on what new trends in the data that they would like to test. Nonetheless, the access to large datasets fake or real was seen as too cumbersome to allocate resource to initially establish not only the actual dataset to use but also to either more the data to a location via Hadoop or allow linkage (via configurations and possible bolt-on middleware). None of the VIHA staff recommended other datasets in the Province, Canada or US or abroad. Ministry Health required several levels of protocol and signed forms even before a meeting could be established to ask if there is a large hospital-level database to access. CIHI has data sets when I requested from them but the data was very small and summarized according to a few chronic diseases for public health, pharmaceuticals or for hospitalization over one year and under highlighted occurrences. I2B2 (in Maryland, USA) was recommended when I asked

Cerner Associates and this organization did have data sets but when I received approval the datasets were very small, summarized for public health reporting or snippets of free text from EHR, which is not what I needed to prove over the platform in order to eventually use patient data of the entire hospital system.

There were no "land mines" in my quest for using real data outright but when engaged with all stakeholders the time to come to a full discussion was taking too long and there were a few conflicts with the resource allocation of labour from teams to assist in the data migration and the actual data that physicians and nurses want to use. The data warehouse team for example knows the data structure and storage inside and out, as well as the database administrators and health informatics architects, but ideas of the types of data to include in the large dataset finding new trends was seen as either too complicated or difficult to actually find true trends. Additionally, the middleware and possible bolt-ons via application platform services to the data warehouse can be done as a traditional extract, transfer, and load (ETL) and then Hadoop can distribute a known location to WestGrid. However, externally distributing the data, according to privacy/security teams interviewed would be difficult to get approval. Therefore, my experiences and ongoing future work in big data of hospital systems will have to get approval from privacy/security but more importantly to establish experts and resources in the initial data migration of real data to a disclosed location before the BDA platform can be used.

In the near future, this study will have ten billion patient records generated for simulation and to distribute the data to over 100 nodes. At this scale, the performance of the system is hoped to be comparable with Yahoo's Hadoop hammer cluster (~3800 nodes) with 1 TB of data requiring only 62 seconds. Srirama, Jakovits, and Vainikko (2012) stated that MapReduce could solve most of the world's scientific problems. Therefore, the first future work to set out to accomplish is increase the number of nodes with Hadoop's HDFS and test MapReduce with other technologies to compare and improve performance and break/fixes to establishing accurate health informatics to solve problems in healthcare.

In the last decade, the MapReduce framework has represented the de facto standard of big data technologies. Since this study showed the high importance of MapReduce to highly integrate technology with health informatics directly related to healthcare application, future work and customization of MapReduce is required. This need to coincide with the capacity to use the platform with complex data models of hospital systems is the key to the success, as well as performance benchmarking of the system to have velocity and veracity over the variety of patient data at volumes that are no longer unimaginable or unmanageable.

With the increasing amount of packages available for Hadoop in open-source software (that is the Hadoop project and associated software projects) more usability testing of tools and their visualization success is vital to the future of Health Big Data. Other possibilities include Hive, Pig and Mahout. Hive (Hive, 2016) is a data warehouse framework developed for Facebook; it is built on top of Hadoop with MapReduce and used for ad hoc querying only (not real-time) for complex analyses. Pig (Pig, 2016) is a high-level data-flow language (Pig Latin) and execution framework whose compiler

produces sequences of MapReduce programs within Hadoop. Pig is designed for batch processing of large volumes of data but cannot perform real-time queries (Norberg, Bhatia, Wang, & Wang, 2013). Mahout (Mahout, 2016) is an Apache project for building scalable machine learning libraries, with most algorithms built on top of Hadoop, but has limitation in its performance due to poor configurations of MapReduce (Yu, et al., 2016). Hadoop R, in particular, provides a rich set of built-in as well as extended functions for statistical, machine learning, and visualization tasks such as: data extraction, data cleaning, data loading, data transformation, statistical analysis, predictive modeling, and data visualization. There are a large number of developers working on additional libraries for Hadoop like Lucene and Blur (Seo et al., 2016). More investigation of this study's different libraries of a variety of packages offered in Hadoop's ecosystem (many of have not been used in healthcare applications) is crucial to ascertaining the best possible BDA platform.

There is a need to further explore with the impact big data technologies on the patient data models of hospital systems. Additionally, using other forms of data of free text and digital needs to be further investigated. Moreover, it was initially set out to test security and privacy of the interactive and functional BDA platform. However, due to the limitations of MapReduce, it was determined that it Java code would remain as is and not to add encrypted patient identifiers for personal health number, medical record number, and date of birth. Tang, Fan, Wang, and Yao (2014) have implemented advanced indexed data of extra-large data sets with good performance after major adjustment to MapReduce programming. Further investigations need to not only test the use of MapReduce to encrypt the data distributed in real-time (with other possible customization to improve access to memory caching if file ingestions are to be implemented iteratively), but also to test querying the data afterwards.

Use cases also need to be established for end user computing of the BDA platform, as there as significant technical and usability differences of Spark and Drill. Scott (2015) highlights a current showdown between Apache Spark and Drill as Drill is designed to be a distributed SQL parsing to query engine (that uses existing ANSI-SQL types that most organizations already use) and the key takeaway is both can utilize interfaces over JDBC\ODBC and SQL but with Spark it is a general computation engine that offers limited SQL capabilities while Drill is more flexible that can utilize JSON files (while Spark cannot) and can increase privacy/security. Spark can be highly customized over an interface but this coding is in Java, which can introduce a greater risk of reduced performance. Additionally, when testing all tools under use cases, it is highly recommended to use MobaXterm because its interface allows for enhanced functionality compared to Putty and log files can be easily accessed and stored outside of WestGrid for future reference.

Recommendations for future work in order to use real patient data at large volumes of hospital systems are:

(1) Continued rigorous simulation tests with SQL varieties (and parsers) on fake data to fully randomize; to apply enhancements to forming; and, querying emulated

data in order to prove the system is operational and productive (as evidence to allow for use of real data)

(2) Increase the simulation with more nodes and towards fully integrating to the specific data model of the hospital system with multiple layers in its patient encounters

(3) Establish a complete understanding of the influence of the suite of tool of Hadoop's ecosystem and the choice of the Big Data database on the health informatics and data

(4) Understanding that the big data technologies are new (e.g., Spark and Drill established in 2013 and 2015, respectively, while Apache Lucene is newly developed for advanced free text search libraries) and unknown and how to test them to promote their use while finding new trends in the utilization of the tools as they evolve across all sectors

(5) Continually contact the Health Authority experts and apply to use real data to keep them informed as all are interested as resources may become available and/or project delivery offices might have funding

6. Conclusion

This study comprised a constructed framework to form a HBDA platform for health applications to test patient data. The work is important as very few studies have applied big data technologies for healthcare applications. Very few implementation studies even exist that tested a BDA platform applied to health informatics of patient data of a hospital system. And no studies have tested a variety of big data tools in Hadoop's ecosystem of packages, which this study did. Furthermore, the sheer volume of 3 billion indexed and ~9-10 billion generated shows that one platform could not only be operational and productive for one hospital but many and even at a provincial scale. The volume achieved at a productive and operational level can further lead to more rigorous simulation(s), utilizing mock or de-identified data or even real patient data. The health informatics and its representative metadata was formulated in the formation of the NoSQL HBase as well as the ANSI-SQL (Drill) SQL-like (Spark) data queries displayed results of patient data of a hospital system. Few studies have produced a platform of the NoSQL database that tested ANSI-SQL and SQL-like data queries of patient data of a hospital system and this study adds to big data technologies, data simulation and healthcare applications over large volumes. SQL has been used at VIHA and other health authorities for decades and cannot be quickly replaced but variations on SQL can be consumed into the organization and utilized. Hence, this study achieved the top three V's that define Big Data: high performance (velocity) over its generator of detailed data (variety) that formed extremely large volumes (volume) significantly contributed to ongoing development of Information Management and Information Technologies (IMIT) in healthcare.

The platform was successfully implemented and tested for healthcare applications with moderate resources and able to run realistic ANSI-SQL (Drill) and SQL-like (Spark and Phoenix) queries on three billion records and perform interactive analytics and data visualizations. An integrated solution eliminates the need to move data into and out of the

storage system while parallelizing the computation, a problem that is becoming more important due to increasing numbers of sensors and resulting data.

This study highly demonstrated that operational Hadoop clusters broke and this not uncommon (cf. Rabkin & Katz, 2013). However, significance of using the MapReduce programming model on top of the Hadoop distributed processing platform proved a process of large volumes of clinical data can be accomplished. Moreover, there are issues both good and bad identified with all big data technologies integrated with Hadoop that is a significant contribution to the field of Big Data Intelligence. Usability goals based end user computing and leveraging the existing tools from data warehouse at health authority is important to make a stand on to use Drill versus Spark.

Useful knowledge gained from this study included the following challenges and specific objectives:

(1) data aggregation - actual storage doubled compared to what was expected due to HBase key store qualifiers, Spark and Drill had the same procedure to ingest Hadoop file before running SQL queries;
(2) data maintenance – ingestions to the database required continual monitoring and updating versions of Hadoop-MapReduce and HBase with limitations persisting for MapReduce (ultimately Java performance in the Reducer) from one to three billion;
(3) data integration –
    i. combination of ADT and DAD possible with simulated patient data and followed current clinical reporting but required a data model of the row keys and column families and this needs to be further tested;
    ii. study's three billion indexed data at 30TB equalled six times more rows than current production and archived at most health authorities, which is said to be 500 million rows on average for a health authority with up to three billion for the entire Province;
    iii. large volumes at different scales, i.e., hospital, health authority, Province, and multiple Provinces, can be achieved if ADT and DAD can be formed to flat file of *.csv* format;
    iv. data model (with SQL parsers) was only verified via simplified analytical queries of simulated data as a benchmark test, but not fully integrated to a defined patient data model and health informatics.
(4) data analysis – high performance of 3.5 seconds for three billion rows and 90 columns (30TB of distributed files) achieved with increasing complexity of queries with high performance of Drill to run queries and high usability with customized Spark with Jupyter; and
(5) pattern interpretation of application – randomized patterns found via Spark with Jupyter interface; however, health trends cannot be found via the application and further investigation required using Hadoop with Spark's Machine Learning Libraries (MLLib).

References

Agrawal, D., Bernstein, P., Bertino, E., Davidson, S., Dayal, U., Franklin, M., Gehrke, J., Hass, L...etc (2012). *Challenges and Opportunities with Big Data.* Big Data - White Paper, Computing Research Association.

Agrawal, R., & Srikant, R. (1994). Fast algorithms for Mining Association Rules. *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB'94),* San Francisco, USA, pp. 487-99.

Ahmad, F., Chakradhar, S.T., Raghunathan, A., & Vijaykumar, T. (2014). Shufflewatcher: Shuffle-aware scheduling in multi-tenant MapReduce clusters. *Proceedings of the USENIX conference on USENIX Annual Technical Conference. USENIX Association, 2014,* pp. 1–12.

Ahmad, F., Lee, S., Thottethodi, M., & Vijaykumar, T. (2012). *Puma: Purdue MapReduce benchmarks suite.* [https://pdfs.semanticscholar.org/70bd/563d00fcb402eb7d9f251bea544ecb08f213.pdf]

Ahuja, S.P., & Moore, B. (2013). State of Big Data Analysis in the Cloud. *Network and Communication Technologies, Vol. 2, No. 1*, 2013, 62-70.

Alder-Milstein, J., & Jha, N. (2013). A.K. Healthcare's "Big Data" challenge. *The American Journal of Managed Care, 19(7),* 537-560.

Alvaro, A. Cárdenas, A.A., Manadhata, P.K., & Rajan, S.R. (2013). Big Data Analytics for Security. *IEEE Security & Privacy, November/December 2013,* 74-76.

Ameri, P., Schlitter, N., Meyer, J., & Streit, A. (2016). NoWog: A Workload Generator for Database Performance Benchmarking. *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress,* 666-673.

Apache Hadoop (2016). *MapReduce Tutorial with Word Count Configuration Check.* [https://hadoop.apache.org/docs/r2.7.2/hadoop-MapReduce-client/hadoop-MapReduce-client-core/MapReduceTutorial.html] [Accessed January 2014].

Apache Phoenix (2016). *Apache Spark Plugin.* https://phoenix.apache.org/phoenix_spark.html [Accessed August 2016].

Apixio Inc. (2013). *Advanced text mining improves Medicare advantage coding.* Knowledge-driven outcomes. Research Bulletin. [http://www.apixio.com/].

Ashish, N., Biswas, A., Das, S., Nag, S., & Pratap, R. (2013). *The Abzooba smart health informatics platform (SHIP) – From Patient Experiences to Big Data to Insights.* Abzooba Bulletin.

AWS (Amazon Web Cloud), (2016). *Amazon Elastic Compute Cloud (EC2) - Scalable Cloud Hosting.* [http://aws.amazon.com/ec2/].

Baro, E., Degoul, S., Beuscart, R., & Chazard, E. (2015). Toward a literature-drive definition of big data in healthcare. *BioMed Research International. Volume 2015*, Article ID 639021, 9 pages [http://dx.doi.org/10.1155/2015/639021].

Bateman A, & Wood, M. (2009). Cloud computing. *Bioinformatics 25(12),* 1475.

Bellazzi, R., Diomiduous, M., Sarkar, I.N., Takabayashi, K., Ziegler, A., & McCray, A.T. (2011). Data analytics and data mining: current issues in biomedical informatics. *Methods of Information in Medicine, 50(6),* 536-544.

Benaloh J., Chase M., Horvitz E., & Lauter K. (2009). Patient controlled encryption: ensuring privacy of electronic medical records. In: *Proc ACM workshop on cloud computing security,* 103–14.

Brusic, V., & Cao, X. (2010). Data Snapshot: visual analytics provides insight into cancer vaccine clinical trials. *Drug Discovery and Development,* 1-5.

Canada Health Infoway (2013). *Big Data Analytics.* Emerging Technology Series, White Paper (Full Report).

Canada Health Infoway and Health Information Privacy Group (2012). *Privacy and EHR Information Flows in Canada (Version 2.0).* July 31, 2012.

Cascading, (2016). *Cascading - project home page.* [http://www.cascading.org].

Caban, J.J., & Gotz, D. (2015). Visual analytics in healthcare – opportunities and research challenges. *J Am Med Inform Assoc, 22,* 260–262.

Canadian Institute of Health Information (CIHI), (2012). *DAD Abstracting Manual: Province-Specific Information for British Columbia.* Ottawa, ON. CIHI Publishing.

Castellani, B., & Castellani, J. (2003). Data mining: Qualitative Analysis with health informatics data. *Qualitative Health Research, 13,* 1005- 1019.

Chang F., Dean J., Ghemawat S., Hsieh W.C., Wallach D.A., Burrows M., Chandra T., & Fikes A.E. (2006). Bigtable: A distributed storage system for structured data. *Seventh Symposium on Operating System Design and Implementation (ODI) Seattle, WA: Usenix Association,* 205-18.

Chang R.-S., Liao, C.-S., Fan, K.Z., & Wu, C.-M. (2014). Dynamic Deduplication Decision in a Hadoop Distributed File System. *International Journal of Distributed Sensor Networks, Volume 2014,* Article ID 630380, 14 pages. [http://dx.doi.org/10.1155/2014/630380].

Chang Y.-J., Chen C.-C., Ho J.-M., & Chen C.-L. (2012). De Novo Assembly of High-Throughput Sequencing Data with Cloud Computing and New Operations on String Graphs. In: *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference,* 155–161.

Chawla, N.V., & Davis, D.A. (2013). Bringing Big Data to Personalized Healthcare: A Patient-Centered Framework. *J Gen Intern Med 28(Suppl 3),* S660–5.

Chen, Y. Alspaugh, S. Borthakur, D., & Katz, R. (2012). Energy Efficiency for Large-Scale MapReduce Workloads with Significant Interactive Analysis. *Proceedings of the 7th ACM European Conference on Computer Systems (EuroSys 2012), ACM, New York, USA,* 43–56.

Chen, Y., Alspaugh, S., & Katz, R. (2012). Interactive Analytical Processing in Big Data Systems: A Cross-Industry Study of MapReduce Workloads, *Proceedings of the VLDB Endowment 5 (12),* 1802–1813.

Chen, H., Fuller, S.S., Friedman, C., & Hersh, W. (2005). *Knowledge management, data mining, and text mining in medical informatics.* In: H. Chen, Fuller, S.S., Friedman, C., & W. Hersh (Eds.). Medical Informatics: knowledge management and data mining in biomedicine. Springer, pp. 20-40.

Chen, H., & Fu, Z. (2015). Hadoop-based healthcare information system design and wireless security communication implementation. *Mobile Information Systems, Volume 2015,* Article ID 852173, 9 pages. [http://dx.doi.org/10.1155/2015/852173].

Chen, W.-P., Hung, C.-L., Tsai, S.-J.J., & Lin, Y.-L. (2014). Novel and efficient tag SNPs selection algorithms. *Biomed Mater Eng, 24(1),*1383–1389.

Chen, H., Chiang, H.L., & Storey, C. (2012). Business intelligence and analytics: from big data to big impact. *MIS Quarterly, 36(4),* 1-24.

Chen, J., Chen, Y., Du, X., Li, C., Lu, J., Zhao, S., & Zhao, X. (2012). Big data challenge: a data management perspective. *Frontiers of Computer Science, 7(2),* 157-164.

Chen, X., Lu, C-D., & Pattabiraman, K. (2014). Failure analysis of jobs in compute clouds: A google cluster case study. In: *IEEE 25th International Symposium on Software Reliability Engineering (ISSRE),* 167–177.

Chen, M., Mao, S., & Liu, Y. (2014). Big Data: A Survey. *Mobile Netw Appl 19*, 171–209.

Chen, Z., Yang, S., Tan, S., He, L., Yin, H., & Zhang, G. (2015). A new fragment re-allocation strategy for NoSQL database systems. *Front. Comput. Sci., 9(1)*, 111–127

Chen, Q., Zhang, D., Guo, M., Deng, Q., & Guo, S. (2010). Samr: A self- adaptive MapReduce scheduling algorithm in heterogeneous environment. In: *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference,* 2736–2743.

Chrimes, D., Kuo, M-H., Moa, B., & Hu, W. (2016)a. Towards a Real-time Big Data Analytics Platform for Health Applications. *International Journal of Big Data Intelligence.* In Press.

Chrimes, D., Moa, B., Zamani, H., & Kuo, M-H. (2016)b. Interactive Healthcare Big Data Analytics Platform under Simulated Performance. *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress,* 811-818.

Chung, W-C., Lin, H-P., Chen, S-C, Jiang, M-F., & Chung, Y-C. (2014). JackHare: a framework for SQL to NoSQL translation using MapReduce. *Autom Softw Eng 21*, 489–508

Chute, C.G. (2005). *Medical concept representation.* In: H. Chen, Fuller, S.S., Friedman, C., & W. Hersh (Eds.). Medical Informatics: knowledge management and data mining in biomedicine. Springer, pp. 61-73.

Cios, K.J. (2001). *Medical Data Mining and Knowledge Discovery.* Physica-Verlag, Spring-Verlag Company.

Coenen, F. (2004). Data mining: past, present, and Future. *The Knowledge Engineering Review, Vol 0,* 1-14.

Collen, M.F. (2009). *Computer Medical Databases – the First Six Decades (1950-2010).* In: K.J. Hannah, and Ball, M.J. (Eds). Health Informatics. Springer, pp.1-301.

Cloudera, (2016). *Integrating Hive and HBase - Cloudera Developer Center.* [http://http://www.cloudera.com/blog/2010/06/integrating-hive-and-HBase/].

Cumbley, R., & Church, P. (2013). Is Big Data creepy? *Computer Law & Security Review, 29,* 601-609.

Dai, H.J., Wu, C.-Y., Tsai, R.T.-H., & Hsu, W.-L. (2014). *Test Mining in Biomedicine and Healthcare.* In: Li, X., Ng, S.-K., & Wang, J.T.L. (Eds). Part IV: Text Mining and its BioMedical Applications. Biological data mining and its application in healthcare.

Science, Engineering, and Biology Informatics Vol. 8. World Scientific Publishing, Co. Pte. Ltd. Singapore, pp. 325-372 ISBN 978-9-814-55100-7..

Dai, L., Gao, X., Guo, Y., Xiao, J., & Zhang, Z. (2012). Bioinformatics clouds for big data manipulation. *Biology Direct, 7(43),* 1-7.

Das, S., Sismanis, Y., & Beyer, K.S. (2010). Ricardo: Integrating R and Hadoop. *Proceedings of the 2010 ACM SIGMOD/PODS Conference (SIGMOD'10),* 987-998.

Debortoli, S., Müller, O., & vom Brocke, J. (2014). Comparing Business Intelligence and Big Data Skills: A Text Mining Study Using Job Advertisements. *Business & Information Systems Engineering Wirtschaftsinformatik.* doi: 10.1007/s11576-014-0432-4. 5-17. (in English).

Dean, J., & Ghemawat, S. (2004). MapReduce: Simplified data processing on large clusters. *Sixth Symposium on Operating System Design and Implementation*, San Francisco, CA Usenix Association.

Dean, J., & Ghemawa, S. (2004). *MapReduce: Simplified Data Processing on Large Clusters.* OSDI.

Dean, J., & Ghemawt. S. (2010). MapReduce: A Flexible Data Processing Tool. *Communications of the ACM, 53(1),* 72-77.

Díaz-Uriarte, R., & De Andres, S.A. (2006). Gene selection and classification of microarray data using random forest. *BMC Bioinformatics, 7(1),* 3.

Ding, L., & Huang, S. (2012). ComMapReduce: An Improvement of the MapReduce with Lightweight Communication Mechanisms. *LNCS, Vol. 7238,* pp. 303-19, 2012.

Dinu, F., & Ng, T. (2012). Understanding the effects and implications of compute node related failures in Hadoop. *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing, ACM,* 187–198.

Dobre, C., & Xhafa, F. (2014). Parallel Programming Paradigms and Frameworks in Big Data Era. *Int J Parallel Prog, 42,* 710–738

Docherty, A. (2014). Big Data – ethical perspectives. *Anaesthesia 69,* 387–398.

Dufresne, Y., Jeram, S., & Pelletier, A. (2014). The True North Strong and Free Healthcare? Nationalism and Attitudes Towards Private Healthcare Options in Canada. *Canadian Journal of Political Science, 47(3),* 569–595.

Dunning, T., & Friedman, E. (2010). *Real-World Hadoop.* O'Reilly Publishing. San Francisco, California. ISBN: 978-1-491-92266-8.

Dunning, T., Friedman, E., Shiran, T., & Nadeau, J. (2016). *Apache-Drill.* O'Reilly Media Publications. San Francisco, CA. 2016.

Dutta, H., & Demme, J. (2011). *Distributed Storage of Large Scale Multidimensional EEG Data using Hadoop/HBase.* Grid and Cloud Database Management, New York City: Springer.

Dhyani, B, & Barthwal, A. (2014). Big Data Analytics using Hadoop. *International Journal of Computer Applications (0975 – 8887) Volume 108 – No 12, December 2014,* 1-5.

Ekane, J., & Fox, G. (2008). MapReduce for data intensive scientific analyses. *IEEE Fourth International Conference on Science, E-Science '08,* 277-84.

Elsayed, T., Lin, J., & Oard, W. (2009). Pairwise Document Similarity in Large Collections with MapReduce. *32nd Intl. ACM Conf. on Research & Development.*

Erdmann J. (2013). As Personal Genomes Join Big Data Will Privacy and Access Shrink? *Chemistry & Biology, 20(1),* 1-2.

Faghri, F., Bazarbayev, S., Overholt, M., Farivar, R., Campbell, R.H., & Sanders, W.H. (2012). Failure scenario as a service (fsaas) for Hadoop clusters. *Proceedings of the Workshop on Secure and Dependable Middleware for Cloud Monitoring and Management, ACM,* 1–6.

Fasano, P. (2013). *Transforming Health Care – the financial impact of technology, electronic tools, and data mining*. John Wiley and Sons, Inc.

Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996). *From data mining to knowledge discovery: an overview*. In: G. Fayyad, Piatestky-Shapiro, G., and P. Smyth (Eds), Advances in Knowledge Discovery and Data Mining. Menlo Park, CA: AAAI Press/MIT Press, pp.21-34.

Ferguson, M. (2012). *Architecting a Big Data Platform for Analytics*. White Paper. Intelligent Business Strategies. IBM.

Fisher, D., DeLine, R., & Czerwinski, M. (2012). Interactions with big data analytics. *Interactions May and June,* 50-60.

Foster, R. (2014). *Health Care Big Data is a big opportunity to address data overload.* Matchcite, URL: http://www.zdnet.com/blog/health/big-data-meets-medical- analysis-video/500 [accessed 2014-9-28]

Freire, S.M., Teodoro, D., Wei-Kleiner, F., Sundsvall, E., Karlsson, D., & Lambrix, P. (2016). Comparing the Performance of NoSQL Approaches for Managing Archetype-

Based Electronic Health Record Data. *PLoS ONE 11(3),* e0150069. doi: 10.1371/journal.pone.0150069

Frey, L.J. Lenert, L., & Lopez-Campos, G. (2014). EHR Big Data Deep Phenotyping Contribution of the IMIA Genomic Medicine Working Group. *Year Med Inform 2014, 206-11* [https://dx.doi.org/10.15265/IY-2014-0006].

Gantz, J., & Reinsel, D. (2012). The Digital Universe in 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East. *Study report, IDC, December 2012*. URL [www.emc.com/leadership/digital-universe/index.htm.]

Garrison, L.P.Jr. (2013). Universal Health Coverage-Big Thinking versus Big Data. *Value Health 16(1 Suppl),* S1-S3.

Google, (2010). *Google blesses Hadoop with MapReduce patent license.* [http://www.theregister.co.uk/2010/04/27/google_licenses_MapReduce_patent_to_Hadoop/].

Gosain, A., & Chugh, N. (2014). New Design Principles for Effective Knowledge Discovery from Big Data, *International Journal of Computer Applications (0975 – 8887) Volume 96– No.17, June 2014,* 19-24.

Gray, J. (2008). *Hadoop Sorts a Petabyte in 16.25 Hours and a Terabyte in 62 Seconds (using Jim Gray's sort benchmark, on Yahoo's Hammer cluster of ~3800 nodes).* [https://developer.yahoo.com/blogs/Hadoop/Hadoop-sorts-petabyte-16-25-hours-terabyte-62-422.html] [accessed October 7, 2016].

Greeshma, A.L., & Pradeepini, G. (2016). Input split frequent pattern tree using MapReduce paradigm in Hadoop. *Journal of Theoretical and Applied Information Technology, Vol 84 No.2,* 260-271.

Grossglauser, M., & Saner, H. (2014). Data-driven healthcare: from patterns to actions. *European Journal of Preventive Cardiology, Vol. 21(2S),* 14–17.

Grover, M., Malaska, T., Seidman, J., & Shapira, G. (2015). *Hadoop Application Architectures: Designing Real-World Big Data Applications.* O'Reilly Publishing. San Francisco, California. ISBN 978-1491900086.

Hadoop, (2016). *Apache Hadoop.* [http://Hadoop.apache.org.].

Hansen, M.M., Miron-Shatz. T., Lau. A.Y.S., & Paton, C. (2014). Big Data in Science and Healthcare: A Review of Recent Literature and Perspectives. *Yearbook of Medical Informatics, 9(1),* 21-26.

Haux, R. (2010). Medical informatics: past, present, future. Commentary. *International Journal of Medical Informatics 79,* 599-610.

HBase, (2016). *HBase - Apache Software Foundation.* [http://Hadoop.apache.org/HBase/].

Heafield, K. (2008). *Hadoop Design and k-means clustering.* Google presentation. [https://kheafield.com/professional/google/more.pdf].

Henschen, D. (2010). Emerging Options: MapReduce, Hadoop. *Young, But Impressive. Information Week, 24.*

Hempel, S. (2007). *The strange case of the Broad Street pump: John Snow and the mystery of cholera.* Berkeley: University of California Press.

Hive, (2016). *Hive - Apache Software Foundation project home page.* [http://Hadoop.apache.org/hive/].

Hive HBase, (2016). *Hive-HBase Integration project home page.* [http://wiki.apache.org/Hadoop/Hive/HBaseIntegration].

Holmes, A. (2014). *Hadoop in Practice.* Second edition. Manning. ISBN 9781617292224.

Hoyt, R., Linnville, S., Chung, H-M., Hutfless, B., & Rice, C. (2013). Digital family histories for data mining. *Perspectives in Health Information Management, Fall,* 1-13.

Hripcsak, G. & Albers, D.J. (2013). Next-generation phenotyping of electronic health records. *J Am Med Inform Assoc, 20(1),*117-21.

Huang, H., Tata, S., & Prill, R.J. (2013). BlueSNP: R package for highly scalable genome-wide association studies using Hadoop clusters. *Bioinformatics, 29(1),* 135-136.

Hughes, G. (2011). *How big is 'big data' in healthcare?* URL: http://blogs.sas.com/content/ hls/2011/10/21/how-big-is-big-data-in-healthcare/ [accessed 2014-9-26]

Hunter, D.J., Altshuler, D., & Rader, D.J. (2008). From Darwin's finches to canaries in the coal mine--mining the genome for new biology. *New England Journal of Medicine, 26,* 2760-2763.

Informatics Interchange (2012). Information extraction from narrative data. *American Journal of Health-Systems Pharmacy 69(15),* 455-460.

Jee, K., & Kim, G.-H. (2013). Potentiality of Big Data in the Medical Sector: Focus on How to Reshape the Healthcare System. *Health Inform Res, June, 19(2),* 79-85.

Jin, J., Ahn, G.J., & Hu, H. (2011). Covington MJ, Zhang X. Patient-centric authorization framework for electronic healthcare services. *Computer Security, 30(2–3),* 116–27.

Johnson, E.M., & Willey, N.D. (2011). Usability Failures and Healthcare Data Hemorrhages. *IEEE Security & Privacy,* March/April 2011.

Jonas, M., Solangasenathirajan, S., & Hett, D. (2014). *Patient Identification, A Review of the Use of Biometrics in the ICU.* In Annual Update in Intensive Care and Emergency Medicine. New York – USA: Springer; 679–688.

Jorgensen, A., Rowland-Jones, J., Welch, J., Clark, D., Price, C., & Mitchell, B. 2014. *Microsoft Big Data Solutions.* John Wiley & Sons Inc. Indianapolis, IN.

Jurney, R. (2013). *Agile data science: building data analytics applications with Hadoop.* O'Reilly Publications. San Francisco, California.

Karau, H., Konwinski, A., Wendell, P., & Zatharia, M. (2015). *Learning Spark: Lightning-Fast Big Data Analysis.* O'Reilly Media Publications. San Francisco, CA.

Karim, R., Ahmed, C.F., Jeong, B-S., & Choi, H-J. (2013) An Efficient Distributed Programming Model for Mining Useful Patterns in Big Datasets, *IETE Technical Review, 30, 1,* 53-63.

Karim, M.R., & Jeong, B.S. (2011). A Parallel and Distributed Programming Model for Mining Correlated, Associated, Correlated & Independent Patterns in Transactional Databases Using MapReduce on Hadoop, *Proc. of the 6th Intl. Conf. on CUTE,* 271-6.

Karim, M.R., Jeong, B.S., & Choi, H.J. (2012). A MapReduce Framework for Mining Maximal Contiguous Frequent Patterns in Large DNA Sequence Datasets. *IETE Technical Review, Vol. 29, no. 2 (Mar-Apr),* pp. 162-8.

Kayyali, B., Knott, D., & Van Kuiken, S. (2013). *The big-data revolution in US health care: accelerating value and innovation.* McKinsey & Company.

Khan, N., Yaqoob, I., Hashem, I.A.T., Inayat, Z. Ali, W.K.M., Alam, M., Shiraz, M. & Gani, A. (2014). Big Data: Survey, Technologies, Opportunities, and Challenges. *The Scientific World Journal, Volume 2014, Article ID 712826,* 18 pages.

Klein, D., Tran-Gia, P., & Hartmann, M. (2013). Big Data. *Informatik Spektrum 36, 3,* 319-323.

Kohlwey, E., Sussman, A., Trost, J., & Maurer, A. (2011). Leveraging the cloud for big data biometrics: Meeting the performance requirements of the next generation biometric systems. *In Services, 2011 IEEE World Congress IEEE,* 597–601.

Koppel, R., Smith, S., Blythe, J., & Kothari, V. (2015). Workarounds to computer access in healthcare organizations: you want my password or a dead patient? *Stud Health Technol Inform, 208,* 215-20.

Kovalerchuk, B., & Schwig, J. (2004). *Visual and spatial analysis: advances in data mining, reasoning, and problem solving.* Springer. 576 pages.

Kubick, W.R. (2012). Big Data, Information and Meaning. *Applied Clinical Trials, February,* 26-28.

Kuiler, E.W. (2014). From Big Data to Knowledge: An Ontological Approach to Big Data Analytics. *Review of Policy Research, Volume 31,* Number 4 (2014) 10.1111/ropr.12077.

Kumar, S., Henseler, A., & Haukaas, D. (2009). HIPAA's effects on US healthcare, *International Journal of Health Care Quality Assurance, Vol. 22 Issue 2,* pp. 183 – 197.

Kuo, M.H., Kushniruk, A.W., & Borycki, E.M. (2010). Design and implementation of a health data interoperability mediator: studies in health technology and informatics. *Studies in Health Technology and Informatics, 155,* 181-186.

Kuo, M.H., Kushniruk, A., & Borycki, E. (2011). A Comparison of National Health Data Interoperability Approaches in Taiwan, Denmark and Canada. *Electronic Healthcare, 10(2),* 14-25.

Kuo, M.H., Sahama, T., Kushniruk, A.W., Borycki, E.M., & Grunwell, D. (2014). Health Big Data Analytics: Current Perspectives, Challenges and Potential Solutions. *International Journal of Big Data Intelligence, 1(12),* 114–126.

Kuo, M.-H., Chrimes, D., Moa, B., & Hu, X. (2015). Design and Construction of a Big Data Analytics Framework for Health Applications. *IEEE Proceedings International Conference on Smart City/SocialCom/SustainCom together with DataCom 2015 and SC2 2015, Chengdu, China.* 631-636.

Kuziemsky, C.E, Monkman, H., Petersen, C., Weber, J., Borycki, E. M. Adams, S., & Collins, S. (2014). Big Data in Healthcare – Defining the Digital Persona through User Contexts from the Micro to the Macro. *Year Med Inform,* 82-9. [http://dx.doi.org/10.15265/IY-2014-0014].

Lai, W.K., Chen, Y-C., Wu, T-Y., & Obaidat, M.S. (2014). Towards a framework for large-scale multimedia data storage and processing on Hadoop platform. *J Supercomput 68,* 488–507.

Langer, S., & Bartholmai, B. (2011). Imaging informatics: challenges in multi-site imaging trails. *Journal of Digital Imaging, 24(1),* 151-159.

Langkafel, P. (2016). *Big Data in Medical Science and Healthcare Management.: Diagnosis, Therapy, side Effects*. Walter de Gruyter Verlag GmbH, Berlin/Boston.

Lavarc, N., Boanec, M., Pur, A., Cestnik, B., Debeljak, M., & Kobler, A. (2007). Data mining and visualization for decision support and modeling of public health-care resources. *Journal of Biomedical Informatics, 40*, 438-447.

Li, Y.A. (2011). *Medical Data Mining: Improving Information Accessibility using Online Patient Drug Reviews*. Master's Thesis, MIT.

Li, D., Parl, H.W., Ishag, M.I., Batbaatar, E., & Ryu, K.H. (2016). Design and Partial Implementation of Health Care System for Disease Detection and Behavior Analysis by Using DM Techniques. *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress,* 781-786.

Li, X., Ng, S.-K., & Wang, J.T.L. (2014). *Biological data mining and its application in healthcare.* Science, Engineering, and Biology Informatics Vol. 8. World Scientific Publishing, Co. Pte. Ltd. Singapore. ISBN 978-9-814-55100-7.

Lith, A. & Mattson, J. (2010). *Investigating storage solutions for large data: a comparison of well performing and scalable data storage solutions for real time extraction and batch insertion of data*. Master of Science Thesis, Department of Computer Science and Engineering, Chalmers University of Technology, Göteborg, Sweden, 2010.

Liyanage, H., de Lusignan, S., Liaw, S.T., Kuziemsky, C., Mold, F., Krause, P., Fleming, D., & Jones, S. (2014). Big Data Usage Patterns in the Health Care Domain: A Use Case Driven Approach Applied to the Assessment of Vaccination Benefits and Risks. *Yearbook of Medical Informatics, 9(1),* 27-35.

Madsen, L.B. (2014). *Data-Driven healthcare: how analytics and BI are transforming the industry.* Jon Wiley and Sons, Inc. Hobokem, New Jersey, NY.

Mahout, (2016). *Mahout - Apache Software Foundation project home page.* [http://lucene.apache.org/mahout].

Marr, B. (2016). *How Big Data Is Disrupting Law Firms and The Legal Profession.* http://www.forbes.com/sites/bernardmarr/2016/01/20/how-big-data-is-disrupting-law-firms-and-the-legal-profession/#7f737efa5ed6. Robes/Tech #BigData [Accessed January 2016].

Maier, M. (2013). *Towards a Big Data Reference Architecture.* Master's Thesis. Eindhoven University of Technology, Department of Mathematics and Computer Science.

Manyika, J., Chui, M., Bughin, J., Brown, B., Dobbs, R., Roxburgh, C., & Hung, B. (2014). *Big data: The next frontier for innovation, competition, and productivity.* URL: http://www.mckinsey.com/insights/business_technology/big_data_the_next_frontier_for_innovation [accessed 2014-9-26].

MapReduce, (2010). *The First International Workshop on MapReduce and its Applications (MAPREDUCE'10)* - June 22nd, 2010 HPDC'2010, Chicago, IL, USA. [http://graal.ens-lyon.fr/MapReduce/].

Marozzo, F., Talia, D., & Trunfio, P. (2012). P2P-MapReduce: Parallel data processing in dynamic Cloud environments. *Journal of Computer and System Sciences, 78(5),* 1382-1402.

Martin-Sanchez, F, & Verspoor, K. (2014). Big Data in Medicine Is Driving Big Changes. *Yearbook of Medical Informatics 2014, 9(1),* 14-20.

Mayer-Schönberger, V. & Cukie, K. (2013). *Big data: A revolution that will transform how we live, work, and think*. Houghton Mifflin Harcourt. Chapter 2.

McCormick, T.H., Ferrell, R., Karr, A.F., & Ryan, P.B. (2014). Big Data, Big Results: Knowledge Discovery in Output from Large-Scale Analytics. *Statistical Analysis and Data Mining,* 404-412.

McKenna, A., Hanna, M., Banks, E., Sivachenko, A., Cibulskis, K., Kernytsky, A., Garimella, K., Altshuler, D., Gabriel, S., & Daly, M. (2010). The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Res, 20(9),* 1297–1303.

Mehdipour, F., Javadi, B., & Mahanti, A. (2016). FOG-engine: Towards Big Data Analytics in the Fog. *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress*, 640-646.

Melnik, S., Gubarev, A., Long, J.J., Romer, G., Shiv-akumar, S., Tolton, M, & Vassilakis, T. (2010). Dremel: interactive analysis of web-scale datasets, *Proc. VLDB Endow, 3(1–2),* 330–339.

Miller, P.L. (2000). Opportunities at the intersection of bioinformatics and health informatics. *Journal of the American Medical Informatics Association, 7(5),* 431-438.

Miller, H.J., & Han, W. (2009). *Geographic Data Mining and Knowledge Discovery.* CRC Press, Taylor & Francis Group. London.

Miller, N.A., Farrow, E.G., Gibson, M., Willig, L.K., Twist, G., Yoo, B., Marrs, T., Corder, S., Krivohlavek, L., Walter, A., Petrikin, J.E., Saunders, C.J., Thiffault, I., Soden, S.E., Smith, L.D., Dinwiddie, D.L., Herd, S., Cakici, J.A., Catreux, S., Ruehle, M., & Kingsmore, S.F. (2015). A 26-hour system of highly sensitive whole genome sequencing for emergency management of genetic diseases. *Genome Med. Sep 30,7(1),* 100. doi: 10.1186/s13073-015-0221-8.

Mittal, A. (2013). Trustworthiness of Big Data. *International Journal of Computer Applications (0975 – 8887) Volume 80 – No.9, October 2013*, 35-41.

Mohammed, E.A., Far, B.H., & Naugler, C. (2014). Applications of the MapReduce programming framework to clinical big data analysis: current landscape and future trends. *BioData Mining, 7(22),* 23 pages.

Moise, D., Trieu, T.-T.-L., Bouge, L. & Antoniu, G. (2011). Optimizing intermediate data management in MapReduce computations. *Proceedings of the first international workshop on cloud computing platforms, ACM,* 1–7.

Moniruzzaman, A.B.M, & Hossain, S.A. (2013). NoSQL Database: new era of databases for Big Data Analytics – Classification, Characteristics and Comparison. *International Journal of Database Theory and Application, 6(4),* 1-14.

Mudunuri, U.S., Khouja, M., Repetski, S., Venkataraman, G., Che, A., Luke, B.T., Girard, F.P., & Stephens, R.M. (2013). Knowledge and Theme Discovery across Very Large Biological Data Sets Using Distributed Queries: A Prototype Combining Unstructured and Structured Data. *PLoS One 8(12),* e80503.

Nabavinejad, S.M., Goudarzi, M., & Mozaffari, S. (2016). The Memory Challenge in Reduce Phase of MapReduce Applications. *Journal of Latex Class Files, Vol. 14, NO. 8, August 2016…Transactions on Big Data IEEE.*

Nelson, R., & Staggers, N. (2014). *Health Informatics: an interprofessional approach.* Mosby, an imprint of Elsevier Inc. Saint Louis, Mo.

National Health Institute (NHI), (2014). *Big Data to Knowledge (BD2K).* URL: [http://bd2k.nih.gov/#sthash.gdBo8VeR.dpbs].

Nishimura, S., Das, S., Agrawal, D., & Abbadi, A.E. (2012). *MD-HBase: design and implementation of an elastic data infrastructure for cloud-scale location services.* Springer Science+Business Media, LLC.

Ngazimbi, M. (2009). *Data Clustering with Hadoop.* Master's Thesis. Boise State University.

Nguyen, A.V., Wynden, R., & Sun, Y. (2011). HBase, MapReduce, and Integrated Data Visualization for Processing Clinical Signal Data. In: *AAAI Spring Symposium: Computational Physiology 2011*.

Nordberg, H., Bhatia, K., Wang, K., & Wang Z. (2013). BioPig: a Hadoop-based analytic toolkit for large-scale sequence data. *Bioinformatics 29(23),* 3014–3019.

NERSC. (2016). *Cloud Computing at NERSC.* [http://www.nersc.gov/research-and-development/cloud-computing/].

Ogilive, B.W. (2006). *The science of describing.* Natural history in Renaissance Europe. The University of Chicago Press. Chicago, Ill.

O'Sullivan, P., Thompson, G., & Clifford, A. (2014). Applying data models to big data architectures. *IBM J Res & Dev 58(5), Paper 18 September/November,* 1-12.

Pattuk, E., Kantarcioglu, M., Khadilkar, V., Ulusoy, H., & Mehrotra, S. (2013). BigSecret: A secure data management framework for key-value stores. *Tech. Rep.* [http://www.utdallas.edu/~exp111430/techReport.pdf] [Access December 2015].

Pearlstein, J. (2013). Information evolution: big data has arrived at an almost unimaginable scale. *Wired Magazine 04.16.13* [http://www.wired.com/magazine/2013/04/bigdata/]. [accessed on May 5, 2013].

Peek, N., Holmes, J.H., & Sun, J. (2014). Technical Challenges for Big Data in Biomedicine and Health: Data Sources, Infrastructure, and Analytics. *Yearbook of Medical Informatics 9(1),* 42-47.

Pencarrick Hertzman, C., Meagher, N., & McGrail, K.M. (2013). Privacy by Design at Population Data BC: a case study describing the technical, administrative, and physical controls for privacy-sensitive secondary use of personal information for research in the public interest. *Journal of American Medical Informatics Association, 20(1),* 25-8.

Pig, (2016). *Pig - Apache Software Foundation project home page.* [http://pig.apache.org/].

Podolack, I. (2013). Making Sense of Analytics. *Healthcare Information Management & Communications Canada, 27(1),* 15-23.

Quin, H.F., & Li, Z.H. (2013). Research on the method of Big Data Analysis. *Information Technology Journal 12(10),* 1974-1980.

Rabkin, A., & Katz, R.H. (2013). How Hadoop Clusters Break. *IEEE Software, July/August 2013,* 88-95.

Raghupathi, W., & Raghupathi, V. (2014). Big data analytics in healthcare: promise and potential. *Health Information Science and Systems, 2:3,* 10 pages.

Rallapalli, S., Gondkar, R.R., & Ketavarapu, U.P.K. (2016). Impact of processing and analyzing healthcare big data on cloud computing environment by implementing Hadoop cluster. International Conference on Computational Modeling and Security. *Procedia Computer Science 85,* 16-22.

RHIPE, (2016). *RHINE - R and Hadoop Integrated Processing Environment.* [http://www.stat.purdue.edu/~sguha/rhipe/].

Rindflesch, T.C., Fiszman, M., & Libbus, B. (2005). *Semantic interpretation for the biomedical research literature.* In: H. Chen, Fuller, S.S., Friedman, C., & W. Hersh (Eds.), Medical Informatics: knowledge management and data mining in biomedicine (pp. 141-168). Springer.

Ross, M.K., Wei, W., & Ohno-Machado, L. (2014). "Big Data" and the Electronic Health Record. *Yearb Med Inform 2014, 97-104.* http://dx.doi.org/10.15265/IY-2014-0003.

RHadoop and MapR (2014). *Accessing Enterprise-Grade from Hadoop.* URL: [https://github.com/RevolutionAnalytics/RHadoop/wiki] [accessed 2014-9-28]

Ruan, G., Zhang, H., & Plale, B. (2013). Exploiting MapReduce and data compression for data-intensive applications. *Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery, ACM*, 1–8.

Sadasivam, G.S., & Baktavatchalam, G. (2010). A novel approach to multiple sequence alignment using Hadoop data grids. *Proceedings of the 2010 Workshop on Massive Data Analytics on the Cloud, ACM 2010(2).*

Sakr, S., & Elgammal, A. (2016). Towards a comprehensive data analytics framework for smart healthcare services. *Big Data Research 4,* 44-58.

Sakr, S. Liu, A., & Fayoumi, A.G. (2013). The family of MapReduce and large-scale data processing systems, *ACM Comput. Surv. 46(1),* 11.

Sakr, S., Liu, A., Batista, D.M., & Alomari, M. (2011). A survey of large scale data management approaches in cloud environments, *IEEE Commun. Surv. Tutor. 13(3) (2011),* 311–336.

Saunders, C. J., Miller, N. A., Soden, S. E., Dinwiddie, D. L., Noll, A., Alnadi, N. A., et al. (2012). Rapid Whole-Genome Sequencing for Genetic Disease Diagnosis in Neonatal Intensive Care Units. *Science Translational Medicine, 4(154),* 154ra135. doi.org/10.1126/scitranslmed.3004041

Schadt, E.E. (2012). The changing privacy landscape in the era of big data. *Molecular Systems Biology 8(612),* 1-2.

Schadt E.E., Linderman, M.D., Sorenson, J., Lee, L., & Nolan, G.P. (2010). Computational solutions to large-scale data management and analysis. *Nature Reviews 11,* 647-657.

Schilling, P.L., & Bozic, K.J. (2014). The Big To Do About ‘‘Big Data’’ *Clin Orthop Relat Res 472,* 3270–3272 / doi 10.1007/s11999-014-3887-0.

Scott, J. (2015). *Apache Spark vs. Apache Drill. Converge Blog, Powered by MapR.* [https://www.mapr.com/blog/apache-spark-vs-apache-drill] [accessed October 12, 2016].

Seo, W., Kim, N., & Choi, S. (2016). Big Data Framework for Analyzing Patents to Support Strategic R&D Planning. *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress,* 746-753.

Shah, N.H., & Tenenbaum, J.D. (2012). The coming age of data-driven medicine: translational bioinformatics' next frontier. *Journal of the American Medical Informatics Association 19,* e2-e4.

Siemens, G. (2004). *Connectivism: A learning theory for the digital age.* [http://www.elearnspace. org/Articles/connectivism.htm].

Skiba, D.J. (2014). The Connected Age: Big Data & Data Visualization. *Nursing Education Pespectives, Volume 3 5, Number 4,* 267-269.

Spiekermann, S., & Cranor, L.F. (2009). *Engineering privacy. IEEE Transactions on Software Engineering 35(1),* 67-82.

Song, T-M., & Ryu, S. (2015). Big Data Analysis Framework for Healthcare and Social Sectors in Korea. *Healthc Inform Res. 2015 January;21(1),* 3-9. [http://dx.doi.org/10.4258/hir.2015.21.1.3] pISSN 2093-3681 • eISSN 2093-369X.

Spivey, B., & Echeverria, J. (2015). *Hadoop Security: Protecting your big data platform.* O’Reilly Publishing. San Francisco, California. ISBN 978-1-491-90098-9.

Srirama, S.N., Jakovits, P., & Vainikko, E. (2012). Adapting scientific computing problems to clouds using MapReduce. *Future Generation Computer Systems 28(1),* 184-192.

Sun, J. (2013). Scalable RDF store based on HBase and MapReduce. *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on,*

Hangzhou, China, 20-22 Aug, 2010.

Sun, J., & Reddy, C.K. (2013). *Big Data Analytics for Healthcare.* Tutorial presentation at the SIAM International Conference on Data Mining, Austin, TX. [https://www.siam.org/meetings/sdm13/sun.pdf].

Swedlow, J.R., Goldberg, I., Brauner, E., & Sorger, P.K. (2003). Informatics and Quantitative analysis in biological imaging. *Science, New Series 300, 5616,* 100-102.

Tang, Y., Fan, A., Wang, Y., & Yao, Y. (2014). mDHT: a multi-level-indexed DHT algorithm to extra-large-scale data retrieval on HDFS/Hadoop architecture. *Pers Ubiquit Comput 18,* 1835–1844.

Taylor, R.C. (2010). An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics. *BMC Bioinformatics 2010, 11(Suppl 12), S1*.

Tauro, C.J.M., Aravindh, S., & Shreeharsha, A.B. (2012). Comparative Study of the New Generation, Agile, Scalable, High Performance NOSQL Databases. *International Journal of Computer Applications (0975 – 888) Volume 48– No.20, June 2012,* 1-5.

Tien, J.M. (2013). Big Data: unleashing information. *Journal of Systems Science Systems Engineering 22(2),* 127-151.

Tsumoto, S., Hirano, S., Abe, H., & Tsumoto, Y. (2011). Temporal data mining in history data of hospital information systems. *IEEE Xplore,* 2350-2356.

Twist, G.P., Gaedigk, A., Miller, N.A., Farrow, E.G., Willig, L.K., Dinwiddie, D.L., Petrikin, J.E., Soden, S.E., Herd, S., Gibson, M., Cakici, J.A., Riffel, A.K., Leeder, J.S., Dinakarpandian, D., & Kingsmore, S.F. (2016). Constellation: a tool for rapid, automated phenotype assignment of a highly polymorphic pharmacogene, CYP2D6, from whole-genome sequences. *NPJ Genomic Medicine 1,* 15007. doi:10.1038/npjgenmed.2015.7

University of Victoria, (2013). *Information System Services.* http://www.uvic.ca/systems/assets/docs/pdfs/researchcomputing/Information_Systems_Services_for_Research.pdf [Accesssed January 2014].

University of Victoria, (2016). *WestGrid.* [https://rcf.uvic.ca/westgrid.php] [Accesssed January 2014].

Umer, S., Afzal, M., Hussain, M., Latif, K., & Ahmad, H.F. (2012). Autonomous mapping of HL7 RIM and relational database schema. *Information Systems Frontiers, 14,* 5-18.

Uzuner, Ö., South, B., Shen, S., & Duvall, S. (2011). "2010 i2b2/VA Challenge on Concepts, Assertions, and Relations in Clinical Text". *Journal of the American Medical Informatics Association, 18,* 552-556.

Vashishtha, H., & Stroulia, E. (2011). Enhancing Query Support in HBase via an Extended Coprocessors Framework. *Proceedings 4th European Conference, ServiceWave 2011, Poznan, Poland, October 26-28, 2011.*

Wang, Y., Goh, W., Wong, L., & Montana G. (2013). Random forests on Hadoop for genome-wide association studies of multivariate neuroimaging phenotypes. *BMC Bioinformatics 14(16),* 1–15.

Wang, F., Lee, R., Liu, Q., Aji, A., Zhang, X., & Saltz, J. (2011). Hadoop-GIS: A high performance query system for analytical medical imaging with MapReduce. In: *Atlanta – USA: Technical Report, Emory University*, 1–13.

Wang, B., Li, R., & Perrizo, W. (2014). *Big Data Analytics in Bioinformatics and Healthcare.* Medical Information Science Reference - IGI Global 1 edition.

Wang, S., Pandis, I., Wu, C., He, S., Johnson, D., Emam, I., Guitton, F., & Guo, Y. (2014). High dimensional biological data retrieval optimization with NoSQL technology. *BMC Genomics 15(Suppl 8*), S3

Wang, Y., Xu, C., Li, X., & Yu, W. (2013). JVM-bypass for efficient Hadoop shuffling. In: *IEEE 27th International Symposium on Parallel & Distributed Processing (IPDPS*), 569–578.

Wassan, J.T. (2014). Modeling Stack Framework for Accessing Electronic Health Records with Big Data Needs. *International Journal of Computer Applications (0975 – 8887), Volume 106 – No.1,* 37-46.

White, T. (2015). *Hadoop – The Definitive Guide: Storage and analysis at internet scale.* O'Reilly Publishing. San Francisco, California. ISBN 978-1-491-90163-2.

von der Weth, C., & Datta, A. (2012). Multi-term Keyword Search in NoSQL Systems. *IEEE Internet Computing, January/February 2012*, 34-43.

Wickramasinghe, N., Bali, R.K., Lehaney, B., Schaffer, J.L., & Gibbons, M.C. (2009). *Healthcare knowledge management primer.* Routledge Series in Information Systems.

Wigan, M.R., & Clarke, R. (2013). Big Data's unintended consequences. Computer, *IEEE Computer Society,* 46-54.

Win, K.T., Susilo, W., & Mu, Y. (2006). Personal health record systems and their security protection. *Journal of Medical Systems 30(4),* 309–15.

World Health Organization (WHO). (2015). *Atlas of eHealth Country Profiles – The use of eHealth in support of universal health coverage.* WHO Press. WHO Document Production Services. Geneva, Switzerland. ISBN 978-92-4-156521-9.

Xu, J., Shi, M., Chen, C., Zhang, Z., Fu, J., & Liu, C.H. (2016). ZQL: A unified middleware bridging both relational and NoSQL databases. *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress*, 730-737.

Xue, Z., Shen, G., Li, J., Xu, Q., Zhang, Y., & Shao, J. (2012). Compressionaware i/o performance analysis for big data clustering. *Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications, ACM,* 45–52.

Yan, D., Yin, X.S., Lian, C. et al. (2015). Using memory in the right way to accelerate big data processing. *Journal of Computer Science and Technology (1),* 30–41 Jan. 2015. doi: 10.1007/s11390-015-1502-9

Yao, Q., Tian, Y., Li, P-F., Tian, L-L. Qian, Y-M., & Li, J-S. (2015). Design and Development of Medical Big Data Processing System Based on Hadoop. *J Med Syst 39,* 23 doi: 10.1007/s10916-015-0220-8.

Yang, C-T., Liu, J-C., Hsu, W-H., & Chu, W.C-C. (2013). Implementation of data transform method into NoSQL database for healthcare data. 2013 International Conference on Parallel and *Distributed Computing, Applications and Technologies*. doi: 10.1109/PDCAT.2013.38.

Yu, S.C., Kao, Q.-L., & Lee, C.R. (2016). Performance Optimization of the SSVD Collaborative Filtering Algorithm on MapReduce Architectures. *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress*, 612-619.

Yu, W., Wang, Y., Que, X., & Xu, C. (2015). Virtual shuffling for efficient data movement in MapReduce. *IEEE Transactions on Computers, vol. 64, no. 2,* pp. 556–568, 2015.

Zhang, C. (2010). Supporting multi-row distributed transactions with global snapshot isolation using bare-bones HBase. *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on, Waterloo, Canada, 25-28 Oct. 2010.*

Zhang, K., Sun, F., Waterman, M.S., & Chen, T. (2003). Dynamic programming algorithms for haplotype block partitioning: applications to human chromosome 21 haplotype data. *Proceedings of the Seventh Annual International Conference on Research in Computational Molecular Biology, ACM,* 332–340.

Zhou, L., Jong, Z., & Feng, S. (2010). Balanced Parallel FP-Growth with MapReduce. *IEEE Youth Conf. on Information Computing and Telecom (YC-ICT).*

Zhou, H., Lou, J.-G., Zhang, H., Lin, H., Lin, H., & Qin, T. (2015). An empirical study on quality issues of production big data platform. in IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE), vol. 2. IEEE, pp. 17–26.

ZooKeeper, (2016). *ZooKeeper - Apache Software Foundation project home page*. [http://Hadoop.apache.org/ZooKeeper/].