Titre: Title:	A Customizable On-Demand Big Data Health Analytics Platform Using Cloud and Container Technologies
Auteur: Author:	Ghoncheh Fahimimoghaddam
Date:	2021
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Fahimimoghaddam, G. (2021). A Customizable On-Demand Big Data Health Analytics Platform Using Cloud and Container Technologies [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/9964/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/9964/
Directeurs de recherche: Advisors:	Marios-Eleftherios Fokaefs
Programme: Program:	Génie informatique

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

A Customizable On-demand Big Data Health Analytics Platform Using Cloud and Container Technologies

GHONCHEH FAHIMIMOGHADDAM
Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie informatique

Decembre 2021

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

A Customizable On-demand Big Data Health Analytics Platform Using Cloud and Container Technologies

présenté par **Ghoncheh FAHIMIMOGHADDAM**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Cheng JINGHUI , président
Marios-Eleftherios FOKAEFS , membre et directeur de recherche
Bram ADAMS , membre

DEDICATION

To my parents
whom I miss so much
And to all the leaders of peace,
love, equality and freedom

ACKNOWLEDGEMENTS

First and foremost, I'd like to thank Prof. Marios Fokaefs of the Computer Engineering department for serving as my supervisor. When I had a difficulty or a question about writing my thesis, he helped me out and pointed me in the right direction when he felt I required it. Furthermore, he was always encouraging me to finish this thesis on my terms. I am grateful to have had the opportunity to study under his supervision and gain expertise in Big Data, Cloud Computing, and related topics. I'd also like to express my sincere gratitude to the jury members who reviewed my thesis. Finally, I'd like to thank Compute Canada for their assistance in providing free access to their platform.

RÉSUMÉ

Avec des solutions technologiques qui ne cessent de se développer, il devient difficile d'identifier les bons outils pour les systèmes d'information de santé. Chaque outil a ses propres avantages et inconvénients, et nombre d'entre eux possèdent plusieurs applications. De plus, les méthodes et méthodologies traditionnelles deviennent insuffisantes à mesure que nous évoluons vers un traitement distribué et en temps réel, car la technologie mondiale évolue rapidement. Le projet a été réalisé en collaboration avec la Clinique Médecine Urbaine du Quartier Latin (CMU), clinique privée spécialisée dans le traitement des maladies transmissibles et des dépendances. L'objectif de ce projet est de prototyper et valider des outils et des procédés pour une plateforme de science des données adaptative et évolutive qui permettra des analyses efficaces, flexibles et performantes, ainsi que la gestion des modèles d'analyse et d'apprentissage automatique qui permettent une prise de décision basée sur la recherche.
Nous évaluons les outils, les méthodologies et d'autres facteurs utilisés pour créer une plateforme de « Big Data » en analysant les avantages et les inconvénients des solutions informatiques matures, de la gestion et des composants de visualisation des plateformes d'analyse pour les données de santé. Ensuite, nous proposons le développement d'un prototype extensible d'Analytics Sandbox pour les données de santé. Nous nous concentrons sur l'évolutivité et l'analyse d'un projet en utilisant de nouvelles technologies dans cette étude. Les infrastructures Cloud et les bases de données distribuées NoSQL sont utilisées pour stocker les données. En outre, des outils tels que Apache Kafka, Spark, Storm, Zeppelin et Sqoop sont utilisés pour faciliter l'ingestion, le transfert et l'analyse rapides des données, ce qui permet d'obtenir des résultats aussi rapides et précis que possible. Enfin, nous menons quelques expériences pour évaluer deux outils de streaming en termes de latence et d'utilisation des ressources et de charge de travail de clustering K-means pour comparer le débit Hadoop et Spark. Les résultats révèlent que la latence dans Storm est minimale par rapport à Spark et Trident. Trident a une utilisation minimale des ressources, même si sa latence est maximale. Les résultats de l'expérience de clustering K-means ont montré que Spark surpasse Hadoop en termes de débit. En conséquence, il peut être un candidat à des fins d'apprentissage automatique.

ABSTRACT

Identifying the right tools for Health Information Systems might be difficult with an ever-increasing number of possibilities. The various tools have their benefits and downsides, and many of them have many applications. Moreover, traditional methods and methodologies have become obsolete as we transition toward distributed and real-time processing because the world's technology evolves fast. The goal of this project, which is being carried out in collaboration with the Clinique Médecine Urbaine du Quartier Latin (CMU), a private clinic specializing in the treatment of communicable diseases and addiction, is to prototype and validate tools and processes for a modern, adaptive, and scalable data science platform that will enable efficient, flexible, and performant analytics, as well as management of analytics and machine learning models that allow for research-based decision making. We evaluate tools, methodologies, and other factors used to build a big data platform by analyzing the benefits and downsides of mature computing tools, management, and visualization components of data analytics platforms for health data. Then, we propose the development of an extensible prototype of an Analytics Sandbox for health data. Our focus is on the scalability and analytics part of the project by employing novel technologies in this study. This platform offers a lightweight and isolated environment that ensures that the maximum available capacity is used to carry out its operations and allows for more optimum hardware utilization. It also enables autoscaling in both applications and infrastructure via real-time performance measures such as memory and CPU consumption in response to peaks in demand. Furthermore, it will automatically divide computing traffic evenly across the replicated containers in the cluster, ensuring stable deployment. It is self-heal system, which means that it can recover itself automatically in the event of a failure. This environment is fully customizable, and it can include all of the tools required for analytics purposes. We used cloud infrastructures to build the platform and NoSQL distributed databases to store data. In addition, tools including Apache Kafka, Spark, Storm, Zeppelin, and Sqoop are used to facilitate fast data ingestion, transfer, and analytics, resulting in as quick and accurate results as feasible. Finally, We ran some experiments to evaluate three streaming tools in terms of latency and resource usage and K-means clustering workload to compare Hadoop and Spark throughput. The results revealed that the latency in Storm is the minimum in comparison with Spark and Trident. Trident has the minimum use of resources, even though its latency is the maximum. The results of the K-means clustering experiment showed that Spark outperforms Hadoop with respect to throughput. Accordingly, it can be a candidate for machine learning purposes.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

BDA	Big data analytics
BDAH	Big data Analytics in Healthcare
IMP	Intelligent Medical Platform
HCLS	Health Care and Life Sciences
EHR	Electronic Health Record
EMR	Electronic Medical Record
HIS	Hospital Information System
NoSQL	Not only SQL (Non Structured Query Language)
AWS	Amazon Web Services
HDFS	Hadoop Distributed File System
OLAP	Online Analytics Processing
SPS	Stream Processing Systems
ESP	Event-Stream Processing
SPE	Stream Processing Engines
DAG	Directed Acyclic Graph
RDD	Resilient Distributed Dataset
MR	Map Reduce

CHAPTER 1 INTRODUCTION

In the digital era, huge volumes of data become available to decision-makers to be used in data-rich environments. Big data refers to datasets that are not only large, but also diverse and fast, making traditional procedures and tools ineffective. [1] Due to the increasing rise of such data, solutions to handle and extract value and knowledge from these datasets must be investigated. Health is among the most affected sectors in this digital world. Millions of patient records or electronic health records (EHRs)[1] are now easily accessible by medical professionals. This volume of accessible digitized data can bring a lot of opportunities like analyzing the data for the practitioners along with challenges for engineers like choosing the right solution to handle or benefit from that amount of data. [2]

These EHRs can be quickly shared, transferred, and studied, as well as analyzed securely, to facilitate diagnosis and treatment. First and foremost, for certain health issues like contagious diseases like HIV, easy, protected, and efficient access to records is available. Moreover, decision-makers would be able to extract useful information from such a diverse and frequently changing set of data, which would result in lowering healthcare costs, improving quality of care, and decreasing waste and errors. [3] So, this prevention and treatment can be crucial for infectious illnesses. Nowadays, Big data analytics in healthcare (BDAH) is used in the health sector to provide such value by early decision making. [3] Thanks to the ability of BDAH to analyze trends and commonalities across individuals, the doctors are able to come up with improved treatments. [1] It can greatly accelerate research on epidemiology and public health. Big Data technology aids in the collection and consolidation of massive amounts of data from medical sources or EHR, as well as the processing and analysis of that data in real time employing cloud computing, in order to gain a better understanding of the data. As a result, it leads to improved decision making. [4, 5]

On the other hand, this availability of digitized data comes with various risks and challenges. Firstly, the data that becomes available and needs to be stored and processed can be of great volume and variety, which are challenges associated with Big Data and require appropriate hardware and software infrastructure for storage and processing. In addition, health records generally constitute data for which security, privacy and anonymity are of the highest importance, especially when they need to be shared among medical professionals and become accessible to researchers. Another challenge we may encounter in data storage or processing can be performance and scalability. Accessibility, privacy [6], security, usability, implementation costs, transportability, interoperability, data cleaning and maintenance, standardisation are all the issues that must be addressed simultaneously with modern solutions [7] to allow

the efficient storage and analytics of the data, its controlled and secure accessibility and to enable, facilitate and accelerate relevant research.

Health data is used in the decision-making process in the healthcare industry, [1] [3] and this data arrives in a variety of formats, including unstructured and complex data that relational database management systems (RDBMS) are unable to handle effectively. Hence, NoSQL databases which are the current and widespread solution and can be used to tackle data volume and variety by using flexible models and storing data differently but more efficiently. However, we must remember that choosing the appropriate technology for the type and nature of data is always critical. In addition to big data technology, Cloud Computing is a promising technology that can be used to provide on-demand data storage, processing, and analysis. Due to the extreme confidentiality and sensitivity of medical data, the health sector has always been cautious to use public infrastructures [8]. But the usage of cloud infrastructure can greatly improve the scale of managing ever-increasing amounts of data, significantly cut handling and maintenance costs, and boost collaboration and data sharing amongst healthcare groups. We also need a distributed system on a cloud environment to achieve efficient performance and address scalability difficulties.

Data science platforms including different aforementioned technologies built in the cloud environment have a crucial role to play in resolving current and future concerns in the management of enormous amounts of data in healthcare by assisting in the processing of large data quantities, sophisticated system modelling, and obtaining derivations from healthcare data and simulations. As a result, in order to find an efficient solution for the health sector, the need for a thorough review and evaluation of the current solutions and research studies before proposing a solution was felt to be necessary. There is a plethora of solutions, but there is a lack of integrated solutions in the form of data platforms, that include storage, analytics, security and other tools that can facilitate the healthcare industry in handling large volumes of data efficiently and in a cost-effective manner and also to guarantee scalability. The data platforms that already exist lack a certain degree of flexibility. While they may offer a plethora of popular tools, not all of them may be necessary for every type of analytics or data study. Furthermore, these platforms may also lack particular automation concerning the management of the tooling and its infrastructure. In our work, we propose a flexible, personalized, and scalable health analytics infrastructure to address precisely these challenges. In the end, the final analytics sandbox solution will be used to assist researchers better in diagnosing the disease and proposing appropriate treatment options. As we already explained, Big data analytics (BDA) is appearing as a potential subject for extracting meaningful information from massive data volumes and boosting decision-making effectiveness. [7] This work mainly presents the benefits and components of big data analytics in the

healthcare (BDAH) industry and proposes an improved big data healthcare architecture that processes and analyses healthcare data on a large scale in a cloud computing environment benefiting from the state of the art technologies to achieve our goals. In collaboration with the Clinique Médecine Urbaine du Quartier Latin (CMU), a private clinic specialising in the treatment of communicable diseases and addiction in the Quartier Latin, the goal of project is to prototype and validate tools and processes for a modern and adaptive data science platform that will allow for efficient storage and secure access to patient records, flexible and performant analytics, and management of analytics and machine learning models that allow for record analysis for research purposes. We are convinced that new technologies, such as NoSQL database systems and distributed analytics (such as Spark, Storm and kafka, zeppelin), may considerably simplify clinic operations. The intended project is divided in the following technical objectives: In Storage layer, the goal is to design and implement a prototype for a back-end data system that will provide mechanisms for the efficient and effective ingestion of data (including cleaning and preprocessing) and its secure and reliable storage. In the mangement layer, the goal is to design and implement a prototype for a software system that will allow the interaction between analytics and the stored data with the goal to optimize queries and analyses on parts of the data. However, in processing layer, the objective is to design a prototype for an analytics platform that will enable the definition of models and analytical tasks, their management and control of versions, also including a well-designed user interface.

The contributions of this work are categorized into three sections; the first section is to recognize the state of the art technologies and architectures of data platforms for health sectors by studying the relevant literature. The second and main contribution of this thesis is to propose and deploy a customizable on-demand analytics sandbox for health information using big data open source technologies such as Spark, Storm, Zeppelin, and NoSQL technologies such as MongoDB and Cassandra. In other words, this sandbox is which means we can pick any and only the services that we need. Besides, It is lightweight and secure solution. In fact, we only load the analytics tools and the data that we need to work on it. Thanks to Kubernetes which provides a self-managed and scalable environment. It it provides a user-friendly interface from where we can manage our projects and analytics workloads securely. In the last section and as a part of designing the architecture, we designed and conducted experiments to evaluate and compare the performance of some alternative tools for our platform; Spark, Storm and Storm Trident for stream analytics and Spark and Hadoop using HiBench for ML workloads (using K-means as a case study).

CHAPTER 2 LITERATURE REVIEW

2.1 Introduction

Understanding the state of the art and the state of practice architecture reflected by related research works, is critical before proposing or designing a new one, as outlined in the previous section. We reviewed relevant literature intending to determine how BDAH systems are designed and what components and technologies are included in relevant architectures. A second objective is to pinpoint any limitations or omissions in these architectures to guide the focus of our platform. So that we can design new platform with required tools and extend platforms based on what were missing on them.

To sum up, this literature review aims to address the following research questions (RQs):

- RQ1: "What are the key benefits expected from using integrated data science platforms for health?"
- RQ2: "What are the architectures and the components of a data science platform for health?"
- RQ3: "Who are the users and stakeholders of the platform?"
- RQ4: "What methods researchers used to evaluate performance and effectiveness of their proposed platforms for the healthcare industry?"

2.2 Set up of the Literature Review

Based on the study's research questions, we searched for papers with keywords such as data science platform, BDAH, NoSQL databases, Healthcare, health informatics, and data mining, machine learning, decision making. The Engineering Village digital library [9] was the primary source for the identified works. We focused on relatively recent works (after 2015) and excluded works that did not offer the implementation of a prototype or some evaluation.

2.3 Related Work

Various papers have discussed big data applications and BDAH. Among all of those papers, some papers propose frameworks or models, or techniques to deal with complex healthcare

requirements. This section will summarize chosen papers, how they validate their solutions and discuss the papers in terms of the research questions.
Krishnan et al. [7] investigate a few applications of BDAH, as well as their results. In terms of operations, the vast majority of patient data is made available on demand so that doctors can assess how other remedies have fared around the world and use significant findings to assist better decision-making and treatments. The collaboration of BDAH can result in improved service delivery as well as significant cost reductions. Accessibility, usability, privacy, security, transportability of data, implementation costs, interoperability, and standardisation are all issues that must be given sufficient attention.

Benhlima et al. [10] provide an overview of big data and its application in healthcare. It has been discovered that the use of big data platforms and tools is constantly assisting in managing the healthcare industry's accelerated growth of data. They are conducting an empirical study to examine the role of big data in the healthcare sector. It has been evidenced that remarkable work in the healthcare industry has been done using big data.

Galetsi et al. [2] did a structured evaluation of healthcare big data analytics due to the significant increase of publications in the health sector. They emphasize on how big data tools are used to generate organisation values/capabilities using the resource-based perspective theory, and they examine the following through content analysis of the selected publications: the classification of different types of big data in healthcare, the associated analytic techniques, the value provided for stakeholders, and the platforms and tools for dealing with large health data. They use a variety of real-world examples to demonstrate how improvements in healthcare were made feasible. This gives practitioners, policymakers, and academics useful information while also guiding them in the right possible directions for future research. The study reveals the possible data types and analytics techniques and Big data analytics (BDA) derived values. These data types are clinical patient data and sentiment data, administration and treatment costs data, pharmaceutical, R&D data. The techniques for data analytics may include modeling, simulation , machine learning, visualization, data mining, statistics, web mining, optimization, text mining, prediction analysis, social network analysis. Using BDA can add values and capabilities to the industry such as better diagnosis for personalized healthcare, automated decision algorithm, new business models and services, enhanced experimentation, sharing of information, data transparency, Identification of at-risk populations, segmentation of customized interventions, reducing costs, protecting privacy.

Chauhan et al. [11] highlight recent improvements in big data analytics in the context of healthcare applications. The paper emphasises the rising importance of predictive data analytics by focusing on patient quality care and providing multiple cases. In addition, a

framework with the focus of knowledge discovery is given, as well as a method for providing considerable benefits to computing technology for excellent patient care diagnostics.

Akhtar et al. [5] introduced a platform using a combination of cutting-edge Big Data analytics tools . They proposed the Intelligent Medical Platform (IMP) as a case study for integrating multi-modal data. The focus of their platform was on machine learning solutions that can be applied to a Big Data storage platform. To turn unstructured data into actionable knowledge they employed different machine learning tools and platforms like Apache Mahout, Skytree, Jaspersoft. To evaluate their system's performance and scalability, they used performance tests to validate scalability and also effectiveness of the system with stress testing.

Lin et al. [12] introduced a Big Data analytics solution that reduces the time it takes to process queries in a Hadoop cluster. They adopt Apache Lucene (information retrieval library) and Hadoop in order to implement rapid medical record search for the Home-diagnosis service. They offer a cloud-based architecture for establishing a self-care service called Home diagnostic. A distributed Lucene-based search cluster, in particular, is intended to enable parallel and scalable online medical record retrieval, data analysis, and confidentiality protection services. A Hadoop cluster is used for offline data storage as well as index building to speed up medical record retrieval. They define two test scenarios to evaluate the performance of their proposal.

Ta et al. [13] describe a universal lambda architecture for big data healthcare analytics based on open source platforms such as Hadoop, Apache Storm, Kafka, and NoSQL Cassandra. Stream computing with Kafka, Storm, and NoSQL Cassandra, combined with the Hadoop-HBase system for batch computing, By providing batch and stream computing, an expandable storage solution, and efficient query management, this proposed architecture can facilitate healthcare analytics. This is relevant to ubiquitous health, detection of fraud, pharmacological discoveries, health information systems, and computer-assisted treatment in healthcare.

Johri et al. [14] proposed a Big Data architecture based on the Hadoop platform, Hive to serve SQL queries and transform unstructured data to structured format, and the R programming language to provide statistical findings and visualisations as an application to Big Data potential in the healthcare domain.

Sheeran et al. [15] provided another architecture for big data in healthcare. Health Data Sources, Big Data Technology, Big Data Analytics, and Applications are the four layers that make up this framework. The Health Data Sources layer identifies the numerous sources from which health-related data is derived. The next layer employs Big Data technology to transfer these datasets into software capable of storing enormous amounts of data, paving the

way for data analytics (ex: Hadoop, Spark, NoSQl, Azure Cloud...). The Big Data Analytics Layer is the third layer, and it contains the mechanisms for performing data analytics, such as data warehousing, data mining, and machine learning. The fourth and final layer shows how the end goals of analytics on Health Big Data are implemented.

Maheshwari et al. [16] discuss the implications of current breakthroughs in data analytics and how they might be used in the healthcare industry, with a focus on prediction and visualisation applications. They proposed a system which comprises of multiple tiers including user, data, processing and model. This modular architecture add multiple functionalities to the analytics. This platform is also able to interact with third party systems to transfer data (import and export). In fact, this platform is able to receive data in various formats and pre-process and clean data from any incorrect values, errors, missing vlaues. Once data pre-processing is finished, the existing data with new data will be merged as a part of a training set for a neural network.

Today's widespread adoption of the Internet of Things (IoT), everything around us tends to generate large amounts of data. The need to use cloud infrastructures to help us store this generated data and its processing purposes is increasing. To deal with the issue of long-term batch processing of huge volumes of stored data and real-time analysis introduced by the increasing velocity at which data is generated especially from connected devices (IoT), Taher et al. [17] propose a framework for real-time and batch BDAH based on IoT and the cloud. They put the proposed solution into action using Amazon Web Services (AWS). Furthermore, they use a Raspberry Pi as an IoT device to produce real-time data. Moreover, to validate the solution and report abnormalities, they used a specific application called ECG monitoring. Finally, they test the implementation's performance in terms of response time by changing the velocity and volume of the analyzed data. They also discuss how to provision cloud resources to ensure processing performance in both long-term and real-time scenarios.

Ali et al. [18] proposed another Intelligent Medical Platform (IMP). This platform is a dialogue based medical decision-making solution that delivers medical coaching and recommendation services, based on the methodology of incremental learning. The prototype has a 90% accuracy rate for knowledge acquisition, an 80% user satisfaction percentage for system interaction, and a 95% accuracy rate for system integration with the legacy system.. IMP addresses interactive dialoging, knowledge maintenance, and evidence support to enhance users' confidence. There are four key aspects of the methodology and architecture for the proposed system: 1. knowledge extraction and engineering, 2. dialogue-based interaction, 3. data acquisition, 4. adaptability of user interfaces and system interoperability.

Iyengar et al. [6] discussed important difficulties in cloud-based healthcare analytics systems.

They demonstrate a cloud-based healthcare application system in action. Their system offers advanced privacy protection, which is critical for healthcare applications that deal with sensitive information. At the same time, they enhance performance by performing computations tasks on client systems and caching. They use blockchain to provide secure HCLS (health care and life sciences). They mentioned that there is no cloud-based healthcare data platform that leverages blockchain for the security, data management, or privacy of their platform. However, this platform allows storing data with a variety of privacy requirements. It can gather data from many different sources, including mobile devices such as cellular phones. Users' personal health care data can be collected via mobile devices. One important feature they offer is the ability to perform processing on client devices. Mobile devices or more powerful computers can serve as clients. The ability to process at the client-side allows for the offloading of many computing tasks from the servers.

Mohindra et al. [19] unveiled a cloud platform for the healthcare and life sciences industry sectors. First, they illustrate how data from Health data gateways (including medical records, claims, lab data, genomics data, and medical images), external data sources, and knowledge sources can be merged at cloud scale to accelerate the production of industry-changing insights for better health outcomes. Then they define a cloud architecture and components that allow these solutions and the compliance issues that are so important in healthcare. This architecture includes conceptual components such as health data gateway, exogenous data, visualization and analytics services, data services, health foundation services, and cloud foundation services. Their proposed Health Cloud includes building blocks such as data services, analytics services, cognitive services, and additional services. Each block has its group of services. Their concentration was not tools but the process and mechanisms to create a comprehensive platform.

Landset et al. [20] compare fault-tolerance methodologies, expandability, effectiveness, interface language, and usability of various available processing paradigms as well as the engines that execute them, including MapReduce, Spark, Flink, Storm, and H2O. They next examine machine learning libraries and frameworks such as Mahout, MLlib, and SAMOA and rank them according to speed, scalability, and coverage. Because no single toolset can genuinely represent a one-size-fits-all approach, this paper seeks to make decision-making easier by offering as much information as possible and quantifying the choices. In addition, They discuss various future possibilities for toolkit-based learning throughout this work and recent research in the field using these tools.

Kaur et al. [21] highlight the benefits and components of big data analytics in healthcare (BDAH). They propose a big data healthcare architecture that processes and analyses massive

amounts of healthcare data in a cloud computing environment using Hadoop clusters. Instead of using traditional software systems, healthcare applications are using cloud services for processing of big data. They claim that cloud computing has overcome the majority of health-care data challenges, such as standardisation of health-care record transmission, privacy, and network security. They used Apache Flume and Sqoop to insert a heterogeneous data set of Electronic Health Records (EHR) into the HDFS system. Hbase is used to store multi-structured data. MongoDB is also used for sharding. In this platform Storm is used to perform live streaming. MapReduce model and HIVE are used to examine the data, with machine learning algorithms analysing similar patterns of data. At the end, they used Splunk Hunk to generate reports. However, they did not evaluate their proposed platform.

Chrimes et al. [22] propose a BDAH platform. The framework was used as a solid evidence at Vancouver Island Health Authority (VIHA) where it was used to test simulated patient data from the main hospital system. Finally, they compared the performance of three interactive data analytics solutions; Apache Zeppelin, Jupyter and Apache Drill, to determine which one fits better using Spark for Analytics. Its performance was evaluated in a simulation with data ingestion into the Hadoop file system using several applications of Apache Spark with Apache Zeppelin and Jupyter web-based interfaces, as well as Apache Drill interfaces. The results showed that it took approximately 2 hours to ingest one billion records using Apache Spark. Spark/Zeppelin and Spark/Jupyter were outperformed by Apache Drill. It was, however, limited to running more simplified queries and had very limited visualisations, resulting in poor usability for healthcare. Zeppelin on Spark demonstrated user-friendly interactions for health applications, but it lacked the flexibility of its interface tools and required additional setup time before running queries. Jupyter on Spark provided high performance stacks not only on HBDA platform, but also in tandem to run all queries concurrently with high usability for a variety of reporting requirements by provider.

Visual exploration is common when analysing data where the relationships between variables are not fully understood. However, this is a time-consuming and labor-intensive process, and interesting trends may be missed. Ideally, we would like to concentrate on interesting patterns and locations within the data, such as significant clusters and outliers. Rao et al. [23] propose an open-source platform for interactive Big Data exploration that will be useful in the healthcare industry. In this framework, they used a novel iterative k-means clustering algorithm for identifying clusters in large datasets. This leads to faster visual exploration of new datasets. They applied this algorithm to identify clusters of trends in labour force participation over a 50-year period in order to present a unique perspective by area of medical specialty. In comparison to internal medicine, specialties such as nurse practitioners have seen a significant increase in the number of practitioners.

Mande, et al. [24] mentioned that BDA in healthcare requires a more comprehensive model than traditional data mining; it necessitates an integrated approach to validate novel technologies capable of containing the velocity, veracity, and volume capabilities required to facilitate information innovation across all healthcare data types and domains. They build a distributed platform using Hadoop and R to create an interactive distributed processing environment with simulated patient data to extract side effects in patients by querying entire clinical trial data.

For huge data processing, Marcue et al. [25] give an evaluation of Spark and Flink frameworks. This paper suggested a technique for benchmarking iterative workloads (K-Means and Page Rank) as well as batch workloads (WordCount, Grep, and TeraSort). They looked at four key factors that influence scalability, resource utilization, and execution time.

The performance of Hadoop MapReduce and Spark programming paradigms in terms of compute efficiency are compared by Khan et al. [26]. They use three workloads, such as WordCount, Sort, and PageRank, to evaluate the two programming paradigms, each with a different size of input dataset. Spark outperforms Hadoop MapReduce in every situation, according to the results. As a result, we decided to focus on Spark technologies more than Hadoop for the design of our sandbox analytics.

2.4 Discussion of Research Questions

2.4.1 A Generic Architecture for Big Data Computing in Healthcare Analytics

The paper [13] propose a data platform architecture by providing Batch and stream computing. Other papers using batch processing with Hadoop or Spark. However, The ecosystem's core architecture may be broken down into four layers: storage, processing, and management and visualization. Whereas the study's main focus is on processing layer and visualization tools, it is crucial to comprehend the context in which they might be used in a framework by considering the the entire ecosystem.. Figure 2.1 summarizes different components of a big data platforms and technologies used for each components derived from the literature review. Th visualization layer added to their proposed generic layer system based on other studies. [22]

The data Visualization layer leverages Apache Zeppelin or similar tools to enable users to connect to the semantic framework and create sophisticated visualization that can be arranged on a surface to make reports or pinned to create dashboards shared around the organization. Data processing frameworks responsible for processing and transforming the incoming data make up the top two layers. The intermediate layer's storage systems are in charge of storing

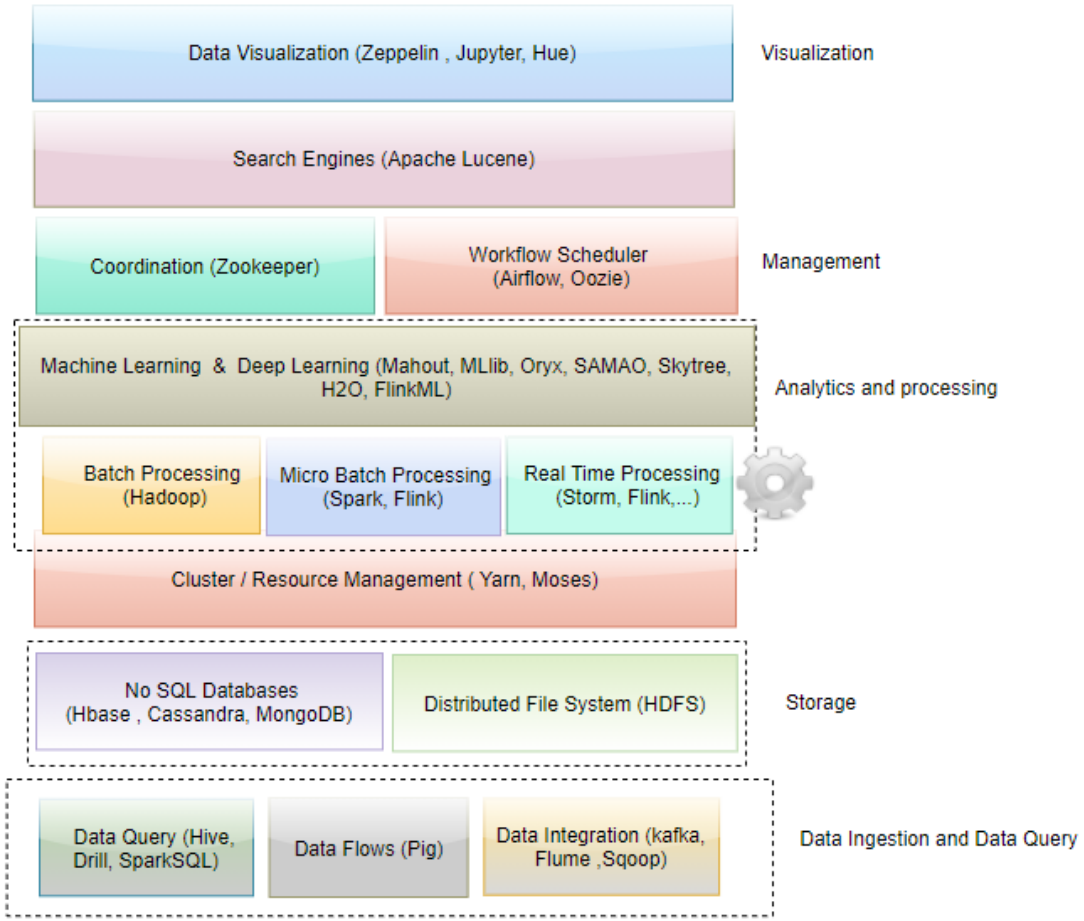


Figure 2.1 Generic Components in Big Data Platform

data in a dispersed form. Resource management systems, which are in charge of scheduling jobs and distributing resources, make up the lowest tier. Each layer's components can usually be swapped out for another on the same layer, affecting the stack's functionality and performance. Figure 2.1 illustrates the layers ecosystem. We will explain the core architecture of the data platform in more depth in sections 3.4 and 3.5.

- **Processing layer:** The actual analysis takes place on the processing layer. It handles processing including machine learning, data transformation and data query. This layer comprises a variety of tools for machine learning and data analysis in addition to the processing engines. This layer also provides data transfer and interaction tools. Data integration solutions like Kafka, Sqoop, and Flume are examples of this. These processing tools are categorized as:
  Analytics Tools: The analytics components is divided into three category based on their processing approach; Batch, Micro-Batch (near Real-time) and Real-time, are re-

sponsible to do analytics in real time and in batch. The concept of real-time processing is to process data as soon as it arrives (Storm). In contrast, batch processing entails storing all received data until a certain amount is collected and then processing it as a batch (Hadoop). The processing data in small groups ("batches") is known as micro-batch processing (Spark).

Data Query Tools: Apache Drill and Hive are both SQL query engines which are different. Apache Drill is a distributed query engine for large-scale datasets. [22]. Besides, Apache Hive is used by Johri et al. [14] and Kaur et al [21]. Hive queries HDFS by converting simple and complex SQL queries into Map Reduce (MR) jobs, and it efficiently uses cluster resources to process historical data. [27]. On the other hand, Drill [22] queries any type of structured and unstructured data sets stored on any file system.

Data Ingestion Tools: This component is used to transfer large amounts of data between Apache Hadoop and data stores in an efficient manner. Apache Sqoop is used to relocate structured data. On the other hand, Apache Flume is used for unstructured/semi-structured data. While Kafka is a distributed publish-subscribe messaging system built on HDFS. On the interface side, query engines like Hive is being used. [21]

Machine Learning Tools: There are machine learning tools working with each Analytics tools such as Mahout, MLib, SAMAO, Oryx, Skytree, H2o, FlinkML. [20] Figure 2.2 also lists various tools and platforms available for machine learning and deep learning purposes.

- **Management layer:** User interaction and high-level organisation tools are included in the management layer. Scheduling, monitoring, coordination, and the user interface are all examples of these. Many of the processing layer's tools, including processing engines such as Pig, Sqoop, Flume and Hive, are managed by Oozie which is a workflow scheduler. It provides the order of operations as well as the collaborates between them. to fulfill tasks in complex workflows that need several jobs and technologies. It also makes it easier to schedule jobs that must run at regular intervals. Zookeeper is a distributed system coordination and synchronisation service. It includes tools for data and protocol coordination, as well as the ability to withstand temporary network outages, which are prevalent in distributed systems. It offers Java and C APIs, as well as Perl, Python, and REST client bindings. [27]

  Resource Management Tool: YARN ("Yet Another Resource Negotiator") [20] is

the open source Hadoop distributed processing framework's resource management and job scheduling technology for Map reduce jobs. it is responsible for managing and monitoring workloads.

Coordination Tools: ZooKeeper is a service for distributed system coordination and synchronisation. It is a necessary tool for Apache storm, kafka, HBase and etc. As a result, it used in all the papers using these technologies.

Workflow Scheduler: Ooziee is a workflow scheduler system that is used to manage jobs. [20]

Search engine: Lin et al. [12] used the Lucene search engine to provide scalable medical record retrieval

- **Visualization layer:** This layer has a responsibility to collect, analyze, and visualize data from various sources to carry out the appropriate monitoring through descriptive and diagnostic analytics, predictive analytics tasks, and prescriptive analytics. Many tools can be used to achieve this goal. Among all of them, we choose Apache Zeppelin, which is an open-source tool. Moreover, It's a Web-based notebook that supports SQL, Scala, Python, R, and other programming languages providing data-driven, interactive data analytics, and collaborative documents.

  Data visualisation Tools: Chrimes et al. [22] used Apache zeppelin, Jupyter. Hue. [20]Hue is a web interface that works with a variety of popular tool, including HBase and HDFS. It includes data visualisation tools and can be used to interact and communicate with Hive, Pig, Sqoop, Zookeeper, and Oozie. [20]

- **Data Storage layer:** This layer is at the bottom of the stack, and it contains the HDFS. A variety of additional distributed data storage alternatives are available that either execute on top of HDFS or operate independently. The HDFS is a file storage system developed for a particular objective rather than a database; therefore, it lacks features found in other data storage solutions. It is well-known for its adaptability and fault tolerance, making it an excellent choice for past data that does not need to be modified or viewed regularly. However, a few drawbacks could affect Hadoop users, particularly those who need quick random reads or writes. Since HDFS is based on a write-once, read-many model, any modifications to a single data point necessitate rewriting the entire file. As a result, many companies opt to incorporate one or more storage systems into their architecture. They are collectively known as NoSQL (Not simply SQL), and since they accept layered, semi-structured, and unstructured data,

non-relational databases, they can be helpful for machine learning tasks. Here are the various types of NoSQL DB's definition: [27]

1. **key-value store:** This is the most basic model among the others, and it is implemented as a large hash table. Each data block is identified by a distinctive key. Voldemort and Redis are two examples of databases based on this paradigm. They are both quick and scalable.
2. **Document stores:** They are layered key-value stores, which means that a critical point to a collection of key-value stores instead of a single value. CouchDB and MongoDB are two examples.
3. **Column-oriented:** In this method, instead of the traditional row/column arrangement, data is kept in columns. Column families are groups of columns. Column-oriented data storage like HBase and Cassandra are two examples.
4. **Graph-based models :** Models based on graphs are designed to work with data that may be depicted as a graph and utilized for activities like analysis of network. With no tables or rows, they are more adaptable than other forms. Titan, Neo4J, and OrientDB are just a few examples.

Data Storage Tools: To store data in a data platform, HDFS, a distributed file storage system, can be used along with other databases types such as relational and Non-relational ones. NoSQL databases can be used for for machine learning purposes because they are able to handle nested, semi-structured, and unstructured data. In studied papers, NoSQL databases (HBase, Cassandra, MongoDB) have been used. Apache HBase is a non-relational database that is used to provide random, real-time read/write access to Big Data. It hosts enormous tables on commodity hardware clusters. Sheeran et al. [15] used Hbase to store results of batch processing with Hadoop. Moreover, they used Cassandra to store the results of stream computing. Kaur et al. [21] used MongoDB in their Hadoop framework.

2.4.2 Evaluation Methods

Each of those studies that propose an architecture used their own way to evaluate their proposed platform. Maheshwari et al. [16] used simulation test to validate accuracy of their machine learning research platform on patients. Chrimes et al. [22] used simulated performance testing to validate visualization tools Spark/Zeppelin, Spark/Jupyter and Apache Drill. Akhtar et al. [5] used stress testing to evaluate the scalability and effectiveness of their proposed platform performance. Lin et al. [12] designed test cases of running examples

with a medical record size of 14GB to test scalability and efficiency of their Lucene-based distributed search cluster in medical record retrieval.

Marcu et al. [25] use a technique for benchmarking batch workloads (WordCount, Grep, and TeraSort) along with iterative workloads (K-Means andPage Rank). They looked at four key factors that influence scalability, resource utilization, and execution time. However, there was no streaming benchmarking of the tools in any of the chosen paper.

Taher et al. [17] measure performance for two type of instances with different cluster sizes and different volumes of training data size during model training (batch processing). They also measure cost of using AWS system while performance testing.

2.4.3 The direct benefits of leveraging integrated data science platforms for health

As we mentioned earlier, the goal of this study is to focus on the analytics section of data platforms for health. So we are going to explain benefits of BDAH understood from the literature review.
In general, The right application of BDAH can result in improved service delivery as well as substantial cost reductions. BDAH simplify the process of storing and retrieving medical data. In hospitals, BDAH incorporating accurate analysis of massive volumes of patient data in accordance with knowledge of treatments can be carried out in order to minimise significant healthcare expenditures and enhance outcomes. Using healthcare analytics in conjunction with effective organisation, streamlining, and analysis of big data will result in faster and accurate diagnosis, fewer preventable errors, and proper treatment, all of which will benefit overall healthcare delivery. In the healthcare area, machine learning and big data analytics are used for risk management, clinical decision support, serious illness diagnosis, and precision medicine.
In overall, in the healthcare domain we can categorize analytics into four types: descriptive, diagnostic, predictive, and prescriptive analytics. [10] A summary overview of any is provided below.(Also shown in figure 2.1)
Descriptive analytics: It is a type of data analysis that is used to describe current situation and event using summary statistics and visualizations, such as histograms, line charts and pie charts.

Diagnostic Analysis: Its goal is to explain how some specific events happened and what variables and factors caused them. Diagnostic analysis, for example, uses approaches like clustering and decision trees to try to figure out why some patients are readmitted on a regular basis.

Predictive Analytics: It indicates one's ability to forecast future events; it also aids in the identification of trends and the estimation of probability for possible future events. Its role can be illustrated by predicting whether or not a patient would develop complications. Machine learning techniques are frequently used to create predictive models.

Prescriptive Analytics: Its purpose is to recommend appropriate behaviours that lead to optimal decision-making. For example, if there is a high risk of a harmful side effect, prescriptive analysis may advocate refusing a treatment. Methods used to undertake prescriptive analytics include decision trees and Monte Carlo simulation. Figure 2.2 depicts the stages of analytics in the healthcare area .

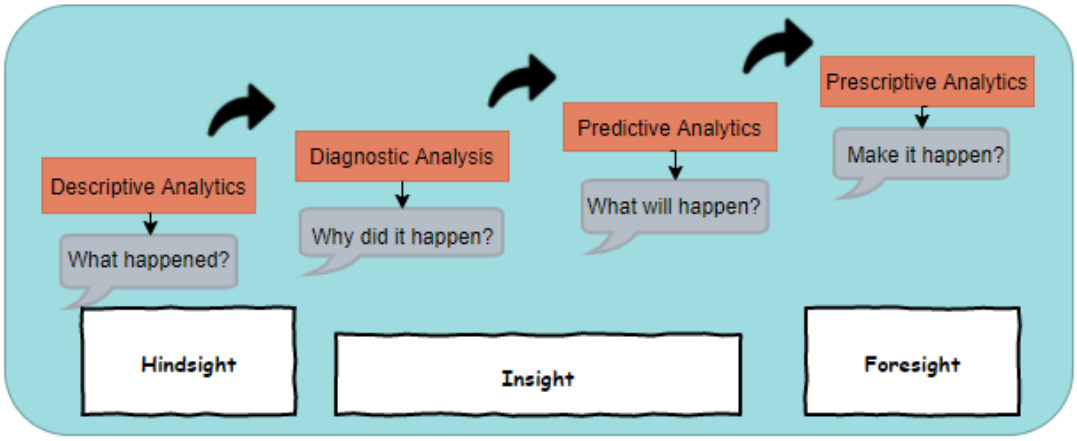


Figure 2.2 Big Data Analytics types in Healthcare

The benefits from using an integrated a BDAH platforms are as follows

1. **Better individualised therapy for Patients:** Analytic solutions for better patient diagnosis lead to more individualised therapeutic schemes or services for users. [2]
2. **Automated algorithms to assist/replace professionals' decision-making:** Analytics can provide diagnostic and remedy/action recommendations using adaptive rules/algorithms for quick categorization of symptoms/medical outcomes and pattern matching. [2]
3. **New solutions, and services and business models:** BDAH allows enterprises to develop innovative services and products. For example, develop new software for data/image analysis, improve existing software, and create totally new business models and ways of reaching out to patients. [7]

4. **Providing opportunities for experimenting, exposing variability, and improving performance:** Analytics facilitates the use of massive datasets for evaluating "what-if" scenarios and assisting performance and decision-making. [7]

5. **Medical data collaboration and coordination:** To improve operational efficiency, BDA can arrange the collection and sharing of information and data analysis across stakeholders. [11]

6. **Increasing data accountability:** BDAH may collect/convert data in a consistent format and handle it the same way every time, saving time, money, and effort while retaining data clarity and quality. [4]

7. **Assessing patient care-risk:** BDAH improves health risk prediction and allows for proactive patient care-risk management. [2]

8. **Delivering better actions by segmenting populations:** BDAH can uncover specific segmentation and tailor products and services to fit the demands of patients or health professionals thanks to its advanced big data exploitation skills. [4]

9. **Reducing costs while increasing quality:** BDAH provides innovative, cost-effective strategies to influence health determinants, with the aim of minimizing costs while maintaining clinical outcomes. [4]

10. **Protecting Patient privacy:** BDAH can identify solutions to protect the privacy of health-related data so that ethical values are upheld and individuals are respected. [6]

Different kinds of users of health information systems can benefit from a data science or BDAH platform such as doctors, caregivers, clinicians, healthcare providers-suppliers, policymakers, payors[1] and patients. [2] [7] A Hospital Information System (HIS) or a data science platform for health would have an impact on nearly every department in the health system. [11]

2.5 Open Issues and Challenges

One of the challenges in a data science platform would be the data standardization and data integration to prepare an integrated dataset [7]. Another challenge is the data platform's scalability; there was no novel solution that proposed a scalable data platform with various scalability mechanisms for health data. Moreover, There are a lot of constantly evolving

[1] payors are insurance companies that pays for an administered medical service

technologies available for building a data platform. So, the need for guidance that compares them and categorizes them for those required to make this platform from scratch for security reasons can be felt. Accessibility, confidentiality, security, usability, operational costs, transportability, interoperability, and standardization are all issues that must be addressed simultaneously. [7] We found that scalability and flexibility of platforms are largely understudied and underdeveloped which is where our work puts the focus on. A benchmarking evaluation of streaming real-time processing tools is also something that was missing. Some studies are available and compare the streaming tools; however, they did not use any evaluation.

CHAPTER 3 SANDBOX ANALYTICS - ARCHITECTURE, IMPLEMENTATION, TECHNOLOGIES

The literature review in previous section has revealed the main components used in BDAH platforms. Based on these components, our purpose in this chapter is to propose the architecture of our platform and outline the technologies that we used to implement each component and the architecture as a whole. According to our literature review, we also found that current solutions have certain limitations with respect to scalability and resource management.

First and foremost, since recent wave of medical record digitalization imposes challenges with respect to volume, variety and velocity in analytics and due to vast, diverse and highly complex ecosystem of the healthcare systems, the main objective of this chapter is to create a unique architecture that is not only scalable but also can process and store all EHRs fast and reliably. Moreover, Security and privacy are also important aspects that must be taken into account for health data which proves analytics in this industry must be done in isolation. Furthermore, all the available data is not necessary for all analytics tasks. So our platform should be able to isolate datasets in an efficient way. Last but not least, due to the plethora of tools and components available for BDAH architecture, we observed that using all tools in the platform can lead to a lack of flexibility which can be a burden to the usability and the performance of the platform. Therefore, we need a customizable solution.

In our architecture, we will not impose any specific technologies and it is not needed to implement all of them at once. In fact, it is not a ready-made solution. On the contrary, our architecture is a flexible, customizable, lightweight, secure and scalable solution; the user of this system can choose the tools they are interested in working with them. Furthermore, thanks to Kubernetes [28], the analytics environment can be quickly launched on-demand. We propose a containerized BDAH. This architecture can be deployed in any cloud environment (like AWS or OpenStack). This chapter will explain (a) how the sandbox architecture looks, (b) what mature technologies are currently using in our architecture, (c) how to use Kubernetes for customizing and starting an environment, and (d) how to achieve autoscaling with Kubernetes.

3.1 Sandbox Architecture

In this section, we present a comprehensive architecture for improving big data computing in healthcare by combining the benefits of micro-batch and stream computing. This architecture

can handle a large-scale data set with minimal latency, as shown in Figure 3.1. This is the basis on which we have built our sandbox environment and our Kubernetes cluster. Sandbox architecture, is based on lambda architecture [29] that can be divided into two computing layers: stream computing, which provides real-time computing through integrated Kafka cluster, Storm cluster, and NoSQL Cassandra, and Micro-batch computing, which provide computing in batches with Spark and stores it in MongoDB database. The serving section will give a common interface for query management and analytics that will assist data analysis.

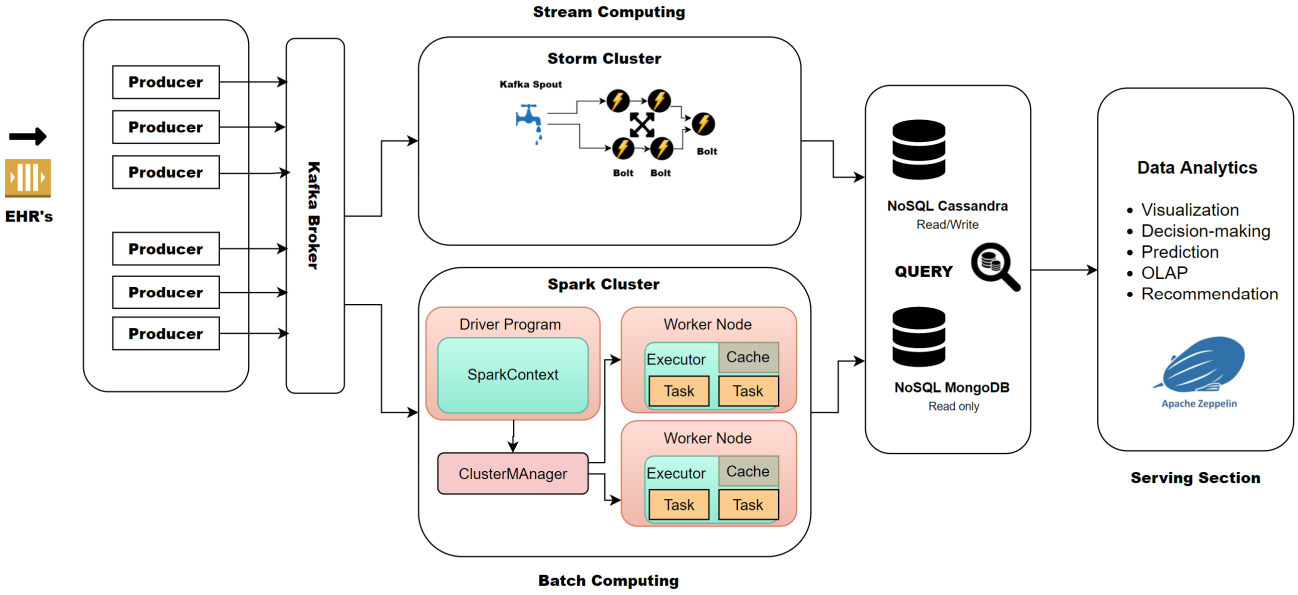


Figure 3.1 BDAH Architecture

Benefiting from this architecture, We will be able to guarantee availability for volatile data. We can store a small number of replicas and do incremental analyses on limited volumes of data. In addition, we perform real-time data analysis (streaming). Moreover, We can guarantee consistency for immutable data. We make data immutable by using the results of real-time analytics. On these data, we do more difficult and expensive analyses (batch processing). In this case, We value accuracy over performance.

An analytics sandbox, is an isolated environment that a user can create and destroy as quickly, and it is does not affect other users of the system. Also, the state of the centralized data is not affected or cause problems to other users. The environment is customized to include all and only the tools that the analyst needs. This way, users are provided with a lightweight environment that is not burdened by unnecessary tools. The Sandbox architecture as it is illustrated in figure 3.1, is a containerized and orchestrated solution build on the

top of Kubernetes cluster. The analytics sandbox solution comprises of the main following components: (Section 3.7 explains all these tools in more details.)
- Apache Zeppelin as a user Interface
- Micro batch and batch processing using Spark
- Stream Processing using Storm
- Kafka messaging system for data integration and native stream management
- MongoDB or Cassandra to store results of analytics (Read-Write/ Write purposes)

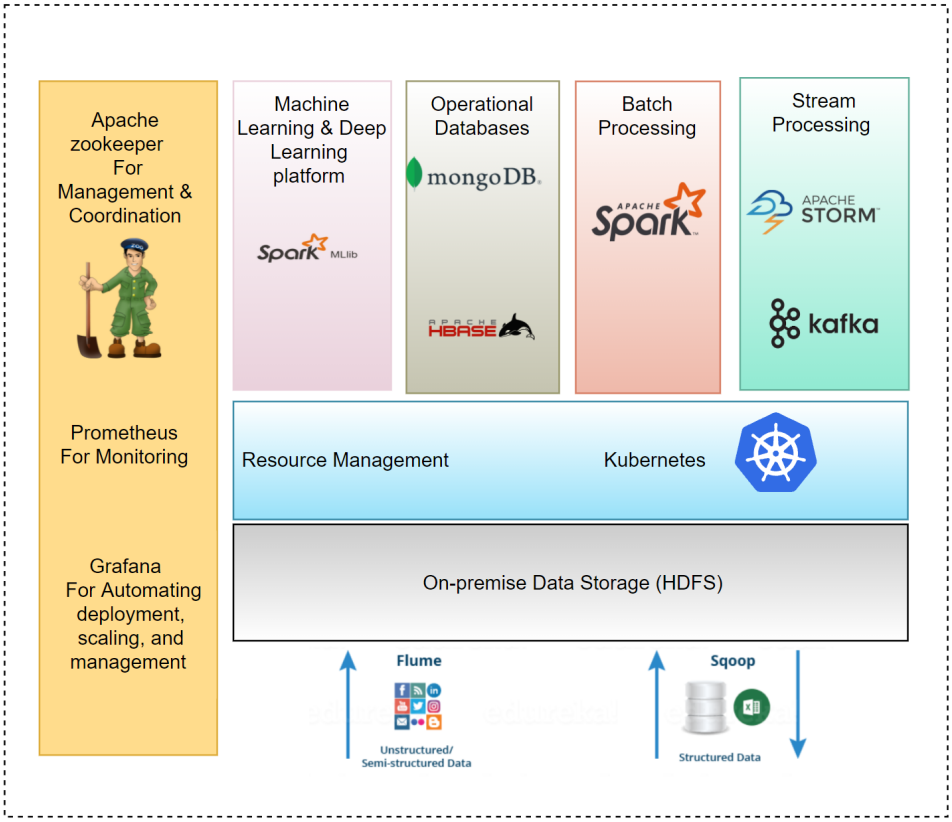


Figure 3.2 Real-time Analytics Sandbox components

Figure 3.2 illustrates Sandbox components. The sandbox is portable concerning the host cloud environment, thanks to Kubernetes and container technology. This sandbox packages big data applications and all their dependencies into Kubernetes containers which would be usable in any other cloud or infrastructure without needing for reconfiguration or repackaging. This ensures portability. Moreover, Using kubernetes technology provides opportunity to create and deploy big data applications in a consistent and repeatable manner. This alleviate deployment of big data systems which are made up of numerous components, each with its own set of dependencies and setup requirements. This will remove the difficulty of mismatched library versions and component compatibility for Spark, Storm, MongoDB

and other similar technologies. The most important aspect of each analytics Sandbox is its scalability. As the resources demand increases, the cluster can respond quickly to peaks in demand. During long-running analytics workloads, particularly significant resources are required, yet utilization of analytics workloads can vary significantly. Dedicated clusters for each deployment incur high expenses due to inefficient resource utilization. However, Sandbox features a set of tools that enable both applications and the infrastructure on which they are hosted to scale up or down based on usage, performance, and various factors. [30] This scalability option conserves costs by scaling down when resources are not needed. Various scaling mechanisms is discussed later in this section. Instead of forcing users to create multiple-segmented clusters, Kubernetes allows users to efficiently share resources so that the same Kubernetes cluster can be used safely for multiple and even concurrent applications. This way of resource management increases utilization while avoiding dependency conflicts and unrestricted resource competition. Namespaces and resource quotas in Kubernetes ensure that diverse workloads share resources fairly, while node selectors and roles can be utilised to isolate resources and access as needed. This will guarantee better resource Management. As a result, adopting an analytics sandbox can facilitate deployment, scalability, and management of big data applications and resources flexibly and reliably. Some benefits of using Kubernetes in our Sandbox would be:

1. Containerize all dependencies as well as applications, eliminates the frequent dependency issues.

2. The Resource Quota and Namespaces features in Kubernetes provides more control over how the applications can use and share system resources.

3. The applications are now portable across hybrid cloud configurations thanks to swappable backend infrastructure.

4. The Kubernetes Role and ClusterRole capabilities allow users to specify fine-grained resource permissions and organise them by API groups.

5. Container images can be tagged for version control, allowing for greater auditing and the ability to rollback unsuccessful deployments.

6. The Kubernetes ecosystem is bursting at the seams with sophisticated open source administration and monitoring add-ons. Some notable examples include Prometheus for time-series data, Fluentd for log aggregation, and Grafana for data visualisation.

7. Helm [31] charts can be used to easily install, manage, and version control packages and their dependencies during the setup process.

3.1.1 User Interface and Zeppelin

Apache Zeppelin [32] is a notebook-based environment that allows users to code while looking at their analyses results simultaneously. Zeppelin through web-based notebooks, make BDAH more straightforward and engaging for healthcare practitioners and scientific researchers for exploration, collaboration, and presentation. It supports numerous language backends and Spark connectivity, handling a wide range of analytic jobs mixing languages within the notebook. It also allows users to connect easily to any JDBC data sources such as Postgresql, Mysql, MariaDB, Redshift, Apache Hive, and many more databases. Underneath is a computational engine known as a kernel, which executes the code in each notebook. In addition, notebooks can make researchers more productive by obscuring cumbersome configurations. Besides, The notebook is easily convertible into a presentation style.

Zeppelin provides numerous interpreters [33] natively, such as Spark, JDBC, Hive, Cassandra, HDFS, Hbase, etc. The Apache Zeppelin Interpreter is a backend for languages. For instance, The Zeppelin Interpreter plug-in enables users to utilize a particular language and data-processing-backend. Furthermore, the setting of the Zeppelin interpreter refers to the configuration of any particular interpreter on the server. Every Interpreter is a member of an InterpreterGroup. InterpreterGroup is a start or stop interpreter unit. Each interpreter is linked to a specific group by default, but the group may comprise many interpreters. The Spark interpreter group, for instance, does include Spark, SparkSQL, pySpark, and the dependency loader. Thus, Zeppelin interpreters within the same group are technically execute in the same JVM. Through Thrift[1], the Interpreter communicates with the Zeppelin engine. As an example, to code with Scala in Zeppelin, the programmer only need a Scala interpreter.

Zeppelin permits its users to create several notebooks, each of which comprises a series of paragraphs. Analysts will build users processing in each paragraph utilizing Zeppelin's languages (SQL, Scala, Shell, Markdown, etc.). It also enables interactive visualization and collaboration with ready-to-use data insights. It allows users to modify the paragraph size (to align numerous paragraphs horizontally), add a header, write code, execute it, and display the results in a table or graph on a web page. It also has a lot more features, such as markdown and JavaScript support (Angular). So analysts can write code, hide it from other analysts, and then build and distribute attractive reports. It also enables users to schedule a job (through cron) to run at a predetermined interval. Finally, once the development is complete, it allows users to switch to report mode and create real-time reports and graphs

[1]The Apache Thrift [34] platform integrates a software stack with a code generation engine to create services that work quickly and smoothly across multiple languages for scalable cross-language service development.

and share them with other users over WebSockets.

In a nutshell, we can sum up Zeppelin as a single interface for all of BDAH requirements.

3.1.2 Batch Analytics

Batch processing [35] is the processing of a large volume of data all at once. The data for a day can easily consist of millions of records and be kept in various ways such as files, records, etc. Generally, the jobs are completed in nonstop, sequential order. Thus, batch data processing is a highly efficient method of parallelized processing on data chunks that have been collected over time. There are some requirements for this type of processing include fault-tolerance, horizontal scalability with increasing load, zero loss of data , and the capability to process all past data or a part of it. [36] Here are some some advantages and disadvantages [37] of using batch processing software:

1. **Advantages:**
   (a) **Cost efficient:** It handles large amounts of data at once, which reduce the costs.
   (b) **Specific Task Processing Duration:** Batch processing systems know how long the job will take to end because it is queued.
   (c) **Independent of any operating system:** To input data into batch systems, no special hardware or system support is required.
   (d) **Less Stress on processor:** Batch systems can work offline, putting less strain on the processor.
   (e) **Handing High Amount of Tasks:** Batch systems can easily handle large amounts of repetitive work.
   (f) **Shared Batch Systems:** Multiple users can share a batch system.
   (g) **Short idle Time:** The batch system has a very short idle time.
   (h) **Offline Features for Scheduling Jobs:** Batch processing jobs can be scheduled to run in system idle time, such as at night or during a free period,
   (i) **Hands-Off Approach:** Using automatic job scheduling in a batch processing system allows hands-off approach.
2. **Disadvantages:**
   (a) **Training for Deployment:** Batch processing systems, necessitate training of what causes a batch to be triggered, how to program processing, and what notifications we expect.

(b) **Debugging:** The complexity of batch systems make debugging difficult.

(c) **Processing delay in case of Error:** If a job takes too long, for instance, if an error occurs, other jobs will be held up for an unknown period of time.

3.1.3 Batch Processing Key Technologies

- Apache Hadoop: It [38] is an open-source platform for storing and processing enormous datasets in a distributed cluster environment. Hadoop allows applications to run on systems with thousands of commodity nodes, handling hundreds of thousands of terabytes of data.

Hadoop can perform advanced processing and analytics on stored information. For example, predictive analysis, data mining, machine learning (ML), and etc. Furthermore, it enables for the division of large data processing in smaller tasks. The small tasks are carried out simultaneously through an algorithm (e.g., MapReduce) and afterwards dispersed across a Hadoop cluster. The Hadoop ecosystem is divided into four major components: [39]

- **HDFS (Hadoop Distributed File System):** A predominant data storage system that handles large data sets on existing hardware. It also has a high data throughput and fault tolerance.
- **YARN (Yet Another Resource Negotiator):** It is a cluster resource manager that schedules tasks and assign resources to applications.
- **Hadoop MapReduce:** It divides large data processing tasks into smaller chunks, distributes the smaller tasks throughout multiple nodes, and afterwards executes each task.
- **Hadoop Common (Hadoop Core):** A collection of shared libraries and utilities on which the other three modules rely.

The application is divided into fragments or blocks that can run on any cluster's nodes. The Hadoop architecture is composed of two types: HDFS, which is the Hadoop Distributed File System and consists of the cluster's Name node and Data nodes, and Map Reduce, which is the Execution Engine and consists of the cluster's Resource Manager and Node Managers. Figure 3.3 depicts a multi-node Hadoop cluster in its entirety. When launching a Hadoop cluster, the HDFS layer launches first, followed by the Map-Reduce layer.
Whereas Apache Hadoop has emerged as one of the best framework for large-scale storage of data and data processing based on the MapReduce model, it does have some significant limitations such as significant disk usage, intercommunication potential is limited. inadequate

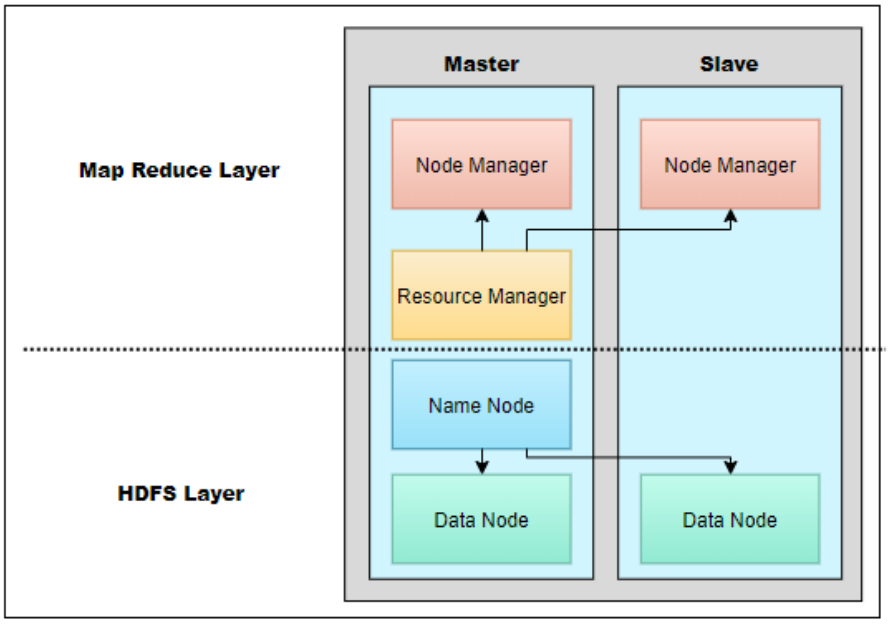


Figure 3.3 Overview of multi-node Hadoop cluster

performance for in-memory computation, unsatisfactory online and iterative computing performance. [36]

- Apache Spark: Apache Spark [36] is a framework that uses in-memory primitives to perform fast distributed computing on Big Data. It is necessary to know the Spark ecosystem. It is comprised of five main modules; however, we only explain modules that are related to batch computing purposes. [39]

- **Spark Core:** The engine that arranges and distributes tasks as well as coordinates input and output (I/O) processes.
- **Machine Learning Library (MLlib):** It includes a collection of scalable machine learning algorithms and techniques which is used for feature-selection and constructing ML pipelines. The primary API for MLlib is DataFrames, that also offers consistency within languages such as Java, Scala, and Python.

RDDs (Resilient Distributed Datasets) are parallel data structures that are immutable. They can save intermediate results in the memory or disk for later use. RDDs are intended to hold massive amounts of data that cannot be accommodated on a single machine; thus, the data must be partitioned across multiple nodes. Spark partitions RDDs automatically and distributes them across multiple nodes. In Spark, a partition is an atomic chunk of data

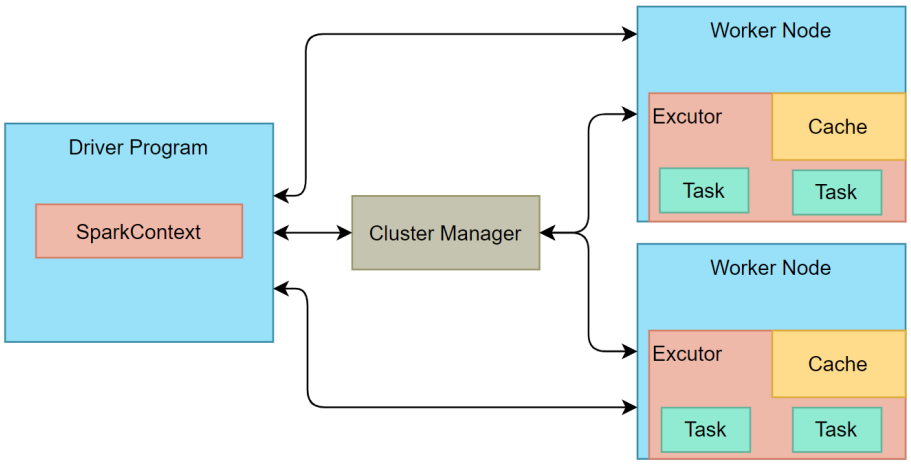


Figure 3.4 Architecture of multi-node Spark cluster

stored on a cluster node. Partitions are the fundamental units of parallelism which can be customized in order to optimize data placement. RDDs are a collection of these partitions. RDD provides API for a variety of data transformations and materialization. [36]

In sum, Spark became an ideal tool for online and iterative processing (particularly for ML algorithms) through loading data in memory which reduces the number of reads and write operations and streamlines the learning process. [20] [36] Moreover, it does not have the limitation of the map-reduce model, such as high disk consumption. The figure 3.4 shows the architecture of multi-node spark cluster.

3.1.4 Stream Analytics

Stream computing is a data processing technology built to handle long-running processes. Apart from batch processing, in which data is finite, which means bound by a start and endpoint in a job, and the tasks end after processing, native streaming is designed to process unbounded data in real-time continuously for long duration like weeks, months, etc. [40] However, a streaming application is challenging to implement and maintain due to its constant need to be up and running. Based on the preceding explanation, it is more evident that there are two categories to divide and implement a Streaming platform. [40]

* **Native Streaming:** Native or real-time streaming, means that every incoming data is handled right away, without having to wait for others. Storm is one example of this type.
* **Micro-batching:** Fast batching or near real-time is another term for it. It means that

incoming records are batched together every few seconds and afterwards in a single mini-batch data will be processed with a few pauses. Spark Streaming and Storm-Trident are two examples.

There are two ways to process data in a stream, Stateless, and Stateful. In the stateless way of data processing, Each incoming record is distinct from the others. There is no connection between records; each one can be processed and saved independently. Stateless processing includes operations such as map[2], filter, join with static data, and so on. In a stateful technique, on the other hand, the processing of an arriving record is dependent on the results of previously processed records. As a result, we must keep track of interim data when processing several records. During processing, every incoming record can read and update this data. In Stateful Processing, this intermediary data is referred to as "State." E.g. StatefulProcessing includes operations such as aggregating the number of records per distinct key, deduplicating records, and so on.

To fully comprehend the strengths and limits of any Streaming frameworks, There are a few key aspects and concepts related to stream processing listed below [40]:

- **Delivery Guarantees:** It refers to the assurance that a specific incoming record in the streaming engine will be processed regardless of the circumstances. It can have approaches such as at-least-once (processed at least once even if there are errors), at-most-once (may not be processed if there are problems), or Exactly-once (processed exactly once even if there are failures). Exactly-once approach is desired, but it is difficult to implement in distributed systems and comes with performance compromises.

- **Fault Tolerance:** In the event of errors such as node failures, network failures, the framework should be capable of recovering and resuming data processing from where it abandoned off. This is accomplished by periodically checkpointing[3] the status of streaming to some storage devices.

- **Status management:** When stateful processing is required, the framework should provide a mechanism for storing and updating state information (e.g., counts of each distinct word viewed in records).

- **Performance:** This comprises latency (the time it takes to process a record), throughput (the number of records processed per second), and scalability. Latency should be

[2]the map() is used to transform one object into other by applying a function.

[3]Checkpointing is a mechanism for storing the state of a computation task so that it can be retrieved and continued at a later time.

kept to a bare possible minimum, while throughput should be maximized. Getting both at the same time is challenging.

- **Advanced Features:** In case of complexity in stream processing requirements, the system may probably need functionalities such as Windowing[4], Watermarks[5], and Event Time Processing. we can mention processing records based on the time they were generated (event time processing).

- **Maturity:** It is desirable from the perspective of adoption which means the framework has already been validated and thoroughly proven at scale by giant organizations. On StackOverflow or similar community, It provesthe likelihood of receiving good community support and assistance. [41]

In summary, both strategies (Native Streaming and Micro-Batching) have their benefits and drawbacks. Native streaming appears natural because each data is processed as soon as received, and permit the framework to accomplish the lowest possible latency. However, because each information must be tracked and checkpointed once processed, it is challenging to provide fault tolerance without sacrificing throughput. State management is also simple because there are long-lasting processes that can readily keep the optimum state. Micro-batching, on the other hand, is different. Because it is effectively a batch, fault tolerance comes free, and throughput is high because processing and checkpointing will be done in a single shot to collect records. There will be some delay, however, and it will not appear to be spontaneous streaming. As a result, maintaining effective state management will be problematic. [40]
Spark and Storm are two mature and widely used stream processing tools (Table 3.1 [41] summarizes various features of Spark and Storm). Hesse et al. [41] mentioned that these tools both have high performance. Moreover, Storm has low latency compared to other available tools. First, however, we need to validate this. Chapter 4 will evaluate Storm and Spark in terms of latency and resource usage.

3.1.5 Stream Processing Key Technologies

Stream processing solutions, also recognised as data stream management systems (DSMS), are built to manage data streams. There are a couple of leading technologies we could use in our proposed platform. Only projects that fall into the category of open-source projects with

[4]Windowing is a method of dividing a data stream into mini-batches or finite streams to apply different transformations.

[5]A watermark is a threshold that indicates when all of the data in a window is expected to arrive.

Features	**Storm**	**Spark**
Scalability	Yes	Yes
High availability	High	High
Latency	Very Low	High
Throughput	Low	High
Cluster Manager	Zookeeper	YARN, Mesos
Event Processing Engine	Yes	Yes
SQL Querying	NO	SparkSQL

Table 3.1 Feature Comparison of stream processing engines (Spark and Storm)

an active group of researchers and developers are considered in this study. In this section, we explain the definition and why every one of these tools has been used.

- Apache Zookeeper: It [42] is a powerful solution for coordinating and managing distributed application configuration and state. ZooKeeper is used by several Apache projects, including stream analytics tools such as Kafka and Storm, etc. Zookeeper is a centralized service that provides configuration information, naming, parallelization, and group services across large distributed systems clusters. The goal is to make these systems easier to manage by improving and ensuring change propagation. With Zookeeper, there is no need to create synchronization services from scratch. Storing status type information in memory on Zookeeper servers provides an architecture for cross-node synchronization for applications. A Zookeeper server preserves a copy of the entire system's state and saves it in local log files. Multiple Zookeeper clusters can manage large big data applications clusters, with a master server synchronizing the top-level servers. An application can use ZooKeeper to produce a znode[6]; a file that stays in memory on the ZooKeeper servers. Any node in the cluster can update the znode, and any node in the cluster can register to be alerted when the znode changes.

- Apache Kafka: Apache Kafka [43] is a high-performance, scalable, and long-lasting publish-subscribe messaging system. Kafka is designed for the delivery of streams. It provides resource isolation between devices or systems that produce and consume data. Kafka is a popular centralized repository for streams, in which events are temporarily stored before being routed anywhere in a data cluster for further processing and analysis. A single Kafka broker can handle large numbers of megabytes of reads and writes per second from millions of users. [44] The Kafka messaging system is used in our analytics sandbox to manage a

[6]Every node in a ZooKeeper tree is refered to as a znode. Znodes keep data that include version numbers for data changes, acl changes, and timestamps. [42]

typical big data gathering in healthcare. A producer sends EMR's to a Kafka topic (messag-

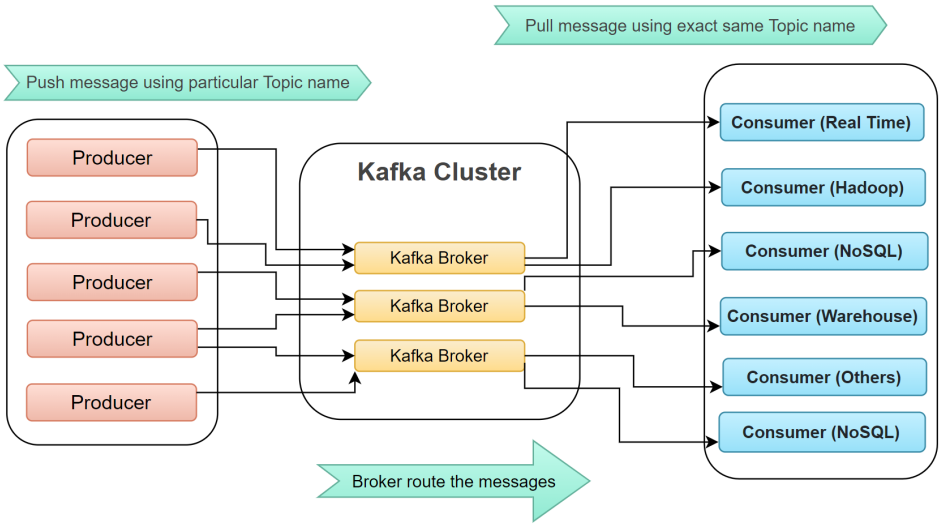


Figure 3.5 Overview of Kafka messaging cluster

ing queue) in its most basic form. Topics are formed on a Kafka broker serving as a Kafka server; Kafka brokers can additionally store messages If needed. Consumers then subscribe to one or more Kafka topics to get messages pulled from the brokers. Offline consumers consume messages and store them in HDFS or a NOSQL database for offline analysis. Real-time consumers can ingest messages, store them in any NoSQL database such as MongoDB or Cassandra, filter them in memory, and send alerts to associated groups. Brokers and consumers, respectively, employ Zookeeper to obtain state information and track message offsets. The procedure is depicted in Figure 3.5.

- Apache Storm: Apache Storm [45] is a distributed real-time computing system that facilitates reliably processing unbounded streams of data. It also can be used with any programming language. It's built to be scalable, dependable, resilient, expandable, efficient, and simple to install and maintain. Storm is now being used in Twitter [46] to do a variety of essential computations at scale and in real-time.

Storm provides a set of generic primitives for real-time big data computing that use Topologies to process data. A Topology is a directed acyclic graph with vertices representing computation components and edges representing the data flow process through computing. The fundamental abstraction in the Storm data processing paradigm is the stream, which is an endless series of Tuples flowing through topologies, as shown in Figure 3.7. A Tuple is nothing more than a collection of named values (key-value pairs). Spouts and Bolts are two

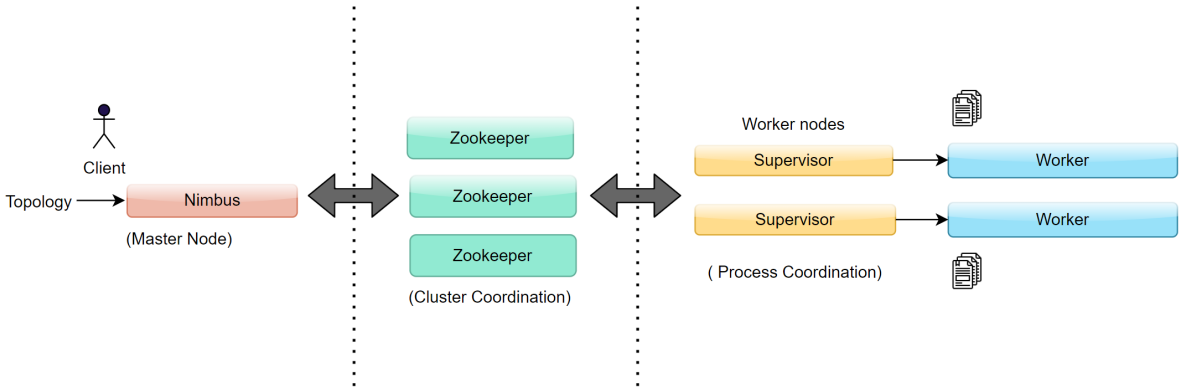


Figure 3.6 Architecture of a Storm System

distinct types of vertices. Spouts are points in the Topology where data streams are inserted. Spouts typically pull data from Kafka message queues. Bolts, on the other hand, process incoming Tuples and pass them downstream, as seen in Figure 3.7.

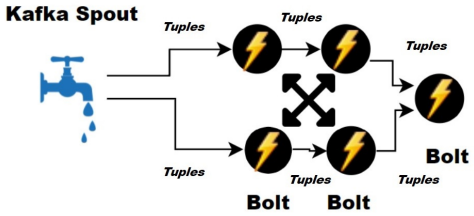


Figure 3.7 Storm Topology Architecture

Bolts are commonly used to filter tuples, perform functions, aggregations, joins, and other processing actions. Similar to distributed computing methods, Storm clusters have a master/slave paradigm. One master node (nimbus) and more worker nodes constitute a Storm cluster (supervisors). In addition to the nimbus and supervisor nodes, Storm requires Apache Zookeeper which acts as storm cluster manager and can include one or more nodes, as seen in Figure 3.6. The master node that runs the Nimbus node, is in charge of distributing and coordinating topology execution in Storm architecture. On each worker node, the Supervisor executes. It accepts tasks from Nimbus and assigns them to upstream workers. Supervisors communicate with Nimbus by exchanging a periodic heartbeat protocol and announcing the Topologies that they are currently executing and any vacancies that may be available to run other topologies. A Zookeeper cluster serves as the hub for all communication between Nimbus and the Supervisors. The Nimbus and Supervisor daemons are built to be fail-fast

and process self-destructs whenever any unprecedented scenario is occurred They are also stateless because the state is stored in a Zookeeper or on a disc. Storm's resiliency is based on this architecture with all of their data stored in Zookeeper or on local storage. Even if the Nimbus service is unavailable, the workers will continue to make progress. In addition, if the workers fail, the supervisors reset them. Some advantages and disadvantages for Storm listed below. [40]

1. **Advantages:**
   (a) Very low latency, genuine streaming, stable and high throughput
   (b) Ideal for simple streaming application cases
2. **Disadvantages:**
   (a) There is no state management.
   (b) There are no advanced capabilities such as event time processing, aggregation, windowing, sessions, watermarks, or other complex features.
   (c) At-least-once delivery guarantee.

- Storm Trident: Trident [47] is a higher-level micro-batching system built on top of Storm. It streamlines the topology-building process and adds higher-level functions like windowing, aggregations, and state management that Storm does not offer natively. In addition to Storm's at most once guarantee, Trident guarantees delivery exactly once. Trident has APIs in Java, Clojure, and Scala.

- Apache Spark: Spark, an Apache top-level project, was formed at the University of California and Berkeley. [48] It supports iterative computation and uses in-memory processing to increase performance and resource usage. Spark framework has completely support the Lambda Architecture (both Batch and Streaming implemented; Batch for correctness, Streaming for Speed) [40]. Batch paradigm of Spark explained in section 3.1.3. In this section, we explain streaming approach of Spark. Spark ecosystem components of streaming are: [39]

- **Spark SQL:** This component of ecosystem gathers structured data information to allow users to optimize structured data processing.
- **Structured Streaming and Spark Streaming [7]:** Both of these technologies enhance the capabilities of stream processing. For a continuous stream, Spark streaming divides

[7]Structured Streaming is a scalable and fault-tolerant stream processing engine that is built on the Spark SQL engine.

data from multiple streaming sources in to the micro-batches. Structured streaming, which is built based on Spark SQL, reduces latency while also simplyfying programming.

Spark streaming employs DStreams to process these data streams into the analytics engine, whereas structured streaming utilizes DataFrames. [49] An RDD sequence represents each DStream. As a result, DStream is nothing more than a collection of RDDs. DFrames, on the other hand, are two-dimensional data structures. It is a distributed collection organized into named columns. Moreover, Datasets are an extended version of the Dataframes API that combines the advantages of RDDs and Datasets. Furthermore, it is a Dataframes extension with additional features such as type-safety and an object-oriented interface. [50]

A more performant version of Spark streaming which is known as structured streaming, includes many useful features such as custom memory management named ,watermarks, event time processing support. [40] Furthermore, Spark still can not be considered a streaming processing tool due to high latency compared to its competitors. The results of benchmarking have also proven this in the Evaluation chapter (chapter 4). However, in its batch computing paradigm, Spark is one of the best available tools. The following are some of the benefits and drawbacks of Spark. [40]

1. **Advantages:**
   (a) Supports Lambda architecture freely.
   (b) High throughput, suitable for a wide range of applications when sub-latency is not required.
   (c) Due to the general micro-batch nature, there is built-in fault tolerance.
   (d) Higher-level APIs are simple to use.
   (e) A large community with a focus on progress
   (f) Exactly-once
2. **Disadvantages:**
   (a) It's not genuine streaming, and it's not designed for low latency.
   (b) There are far too many configuration variables to adjust. It's challenging to get it correctly.

3.1.6 NoSQL Support for Sandbox analytics

In this section, we will compare two NOSQL databases that are beneficial to use for the architecture and explain the benefits of each. Apache MongoDB and Cassandra are two popular

databases that are used in our analytics sandbox. MongoDB is a cross-platform document-oriented database application that is open source. The document data model of MongoDB supports JSON format by default. Moreover, for developers, MongoDB's query is straightforward to learn and apply. [51] It has built-in features, including automatic failover, the ability to scale horizontally and to allocate data to a particular location. [52]. It's a distributed system with a single master that distributes many copies of the data via asynchronous replication for high availability. A MongoDB is a cluster of MongoDB nodes that share the same data. This grouping is referred to as a replicaset in MongoDB documentation. [52]

There are two sorts of data-bearing members in a MongoDB cluster:

1. **Primary Node:** This is the master node, which receives all write requests.
2. **Secondaries:** To preserve an identical data set, the secondaries receive replicated data from the primary.

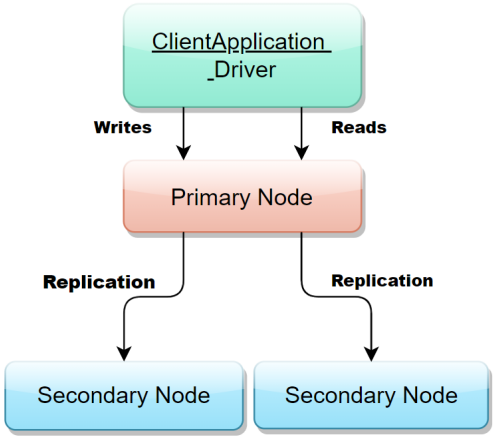


Figure 3.8 A MongoDB cluster

By default, all reads and writes are handled by the primary Node. A MongoDB client can optionally transport some or all reads to secondary members. However, all messages must be written to the primary. If the primary member fails, all writes are halted until one of the secondary members is chosen as the new primary. According to the documentation provided by MongoDB, this operation can take up to 12 seconds to complete. [53] A cluster can be distributed across geographically separate data centers to boost availability. For example, a MongoDB replicaset can have up to 50 members. The Replication process in the MongoDB cluster is shown in the Figure 3.8.

On the other hand, a Cassandra cluster is a collection of connected nodes in a peer-to-peer "share nothing" distributed topology. There is no master node; each node can perform all database operations and serve client requests. Data is partitioned among nodes using a hash of its partitioning key that is consistent. Each node can have one or more partitions, and each partition can contain one or more rows. A partition, on the other hand, can only reside on one node. The replication factor determines the number of copies (replicas) that should be created. The replicas are kept on separate nodes automatically. The coordinator is the node that receives the first request from a client. The coordinator's responsibility is to deliver the request to the nodes with the data and then relay the results back to the coordinator. A coordinator can be any node in the cluster.

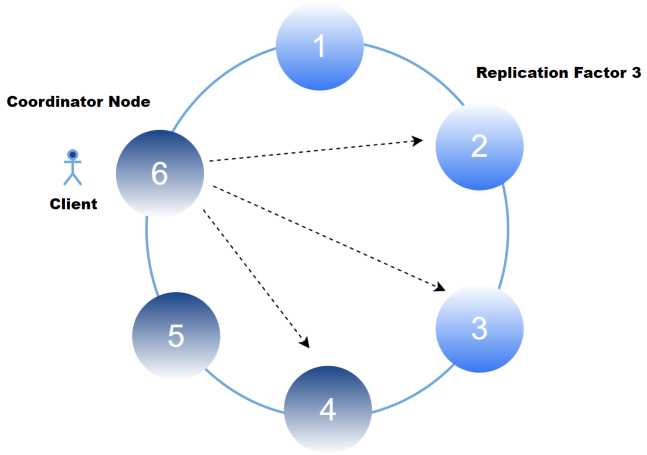


Figure 3.9 Cassandra cluster with coordinator

To choose a database system for analytics sandbox, the first essential factor that comes into mind is the consistency of the data system. MongoDB is a very consistent system. For example, any subsequent read will return the most recent value once a write process is complete. Cassandra is an eventually consistent system by nature. The most recent data gradually becomes available if no more modifications are made after completing the writing process. It's crucial to remember that MongoDB eventually becomes a consistent system when read operations are performed on secondary members. This delay occurs due to replication latency. The longer the delay, the more likely reads will return data that is inconsistent. Furthermore, MongoDB and Cassandra both [54] have "tunable consistency." to suit particular requirements. For example, each reads and writes transaction specifies how many members or replicas are required to recognize a request to be fulfilled. This level is referred to as reading concern or write concern in MongoDB. The consistency level of the operation in Cassandra is the level of compliance. [55] The Figure 3.9 is shown Cassandra

cluster with coordinator.

MongoDB and Cassandra achieve high availability by duplicating data in several places. The greater the number of copies, the greater the availability. MongoDB classifies as a distributed system that prioritizes consistency over availability using the CAP or PACELC [54] theorems, whereas Cassandra classifies as a system that prioritizes availability. These classifications, on the other hand, solely characterize the default behavior of both systems. For example, only if all reads directly to the principal member can MongoDB stay highly consistent. Even when reads are limited to the primary member, read and write concerns can cause inconsistency. Cassandra can also be more consistent by adjusting consistency levels, as long as a loss of availability and increased latency are acceptable trade-offs.

In a nutshell, both Cassandra and MongoDB databases have their own set of benefits and drawbacks. We are that determine what factors in the database are of the highest significance for us. In terms of high availability, Cassandra has the upper hand. We can write to a cluster even if nodes fail because of the highly dispersed architecture. In many use cases, Cassandra's reputation for rapid write and read performance and distributing accurate linear scale performance in a masterless, scale-out design puts it ahead of its NoSQL database competitors. MongoDB, on the other side, thrives at storing unstructured data. It offers a schema-free architecture that allows for efficient caching and logging. Rapid caching and logging operations are essential for real-time analytics and streaming systems. MongoDB is also useful for query speed because it allows secondary indexes. Thus, if the scalability of data activities is essential, Cassandra will be a better fit. Overall, Cassandra performs better with massive data loads due to its ability to support several master nodes in a cluster. At the same time, MongoDB is excellent for workloads involving large amounts of unstructured data. As a result, they each have their advantages based on the data volumes and the implementation needs. MongoDB dramatically simplifies data architecture in an application in a data platform that actuates "transactional, search, mobile, and real-time analytics workloads on any cloud." [56]

3.2 CMU Use Cases

In the CMU case study, and to begin the analytics tasks and visualize it with the zeppelin, we first needed to prepare a spark cluster in three nodes to handle analytics. We needed to install Apache Zeppelin and configure it to connect it to Apache Hadoop, where we store the dataset.

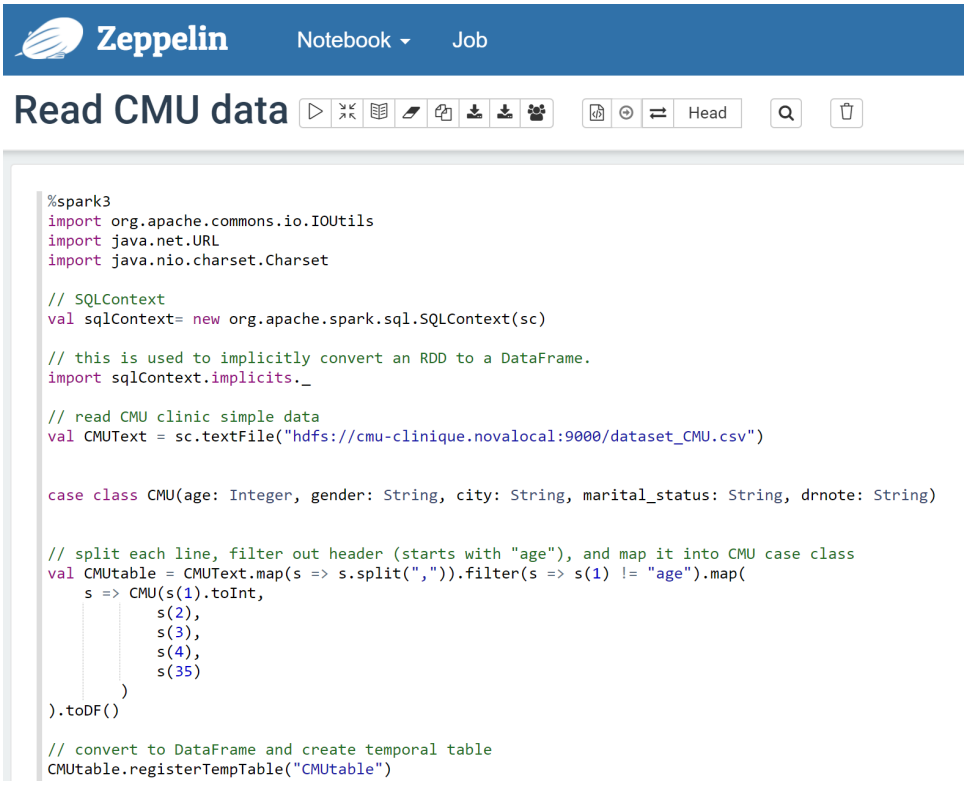


Figure 3.10 SparkScala Code Snippet

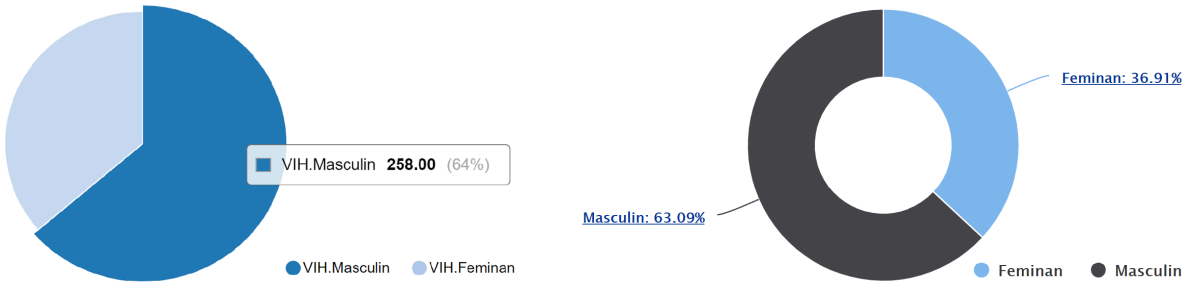
3.2.1 Data Visualization Using Apache Zeppelin

This section describe an example of how we can combine Zeppelin and Spark to run some analytics. As we mentioned in the previous section, Apache Zeppelin is an interactive computational environment. Besides, Apache Spark is an in-memory computational engine with programming APIs that enable data analysts to run streaming, ML, or SQL workloads that require rapid iterative access to datasets. With the aid of Zeppelin researchers will be allowed to build applications to utilize Spark's potential, draw conclusions and improve their analytics workloads within a single, shared dataset in any data store such as HDFS.

To run this simple example, first, we need Apache Zeppelin and a Spark cluster. Then, after the setup of Spark interpreter, we will be able to use notebooks in Zeppelin to run Apache Spark jobs and drawing conclusions from our dataset. We ingest our data from database to HDFS using Sqoop. Our SparkScala code with DataFrames and some filtering read data from HDFS. (Figure 3.10) Then we visualized the results of the analytics using spark.SQL.

The approach we used in this section to analyze health data was descriptive analytics. Using SQL codes we can visualize the results. For instance, we used `SELECT * FROM CMUtable where drnote like '%VIH%'` to start some analytics. This analytics shows us that 64% of

the HIV patients were masculine and only 36% were feminine. It also reveals that 63.09% of male HIV patients were married while 36.91% of female HIV patients were married. (Figure 3.11)



(a) HIV patients Gender percentage

(b) HIV Patients Married percentage

Figure 3.11 Visualization Results

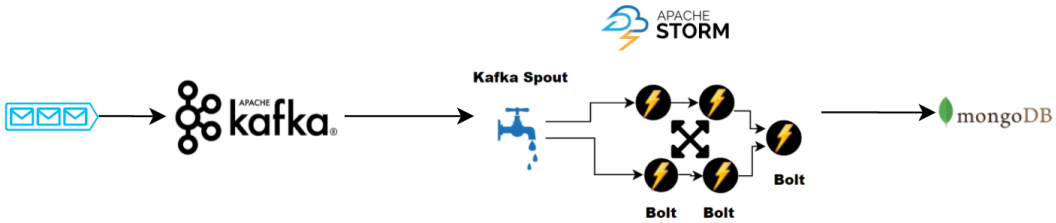
3.2.2 Stream Analytics using Storm, Kafka and MongoDB

This section will briefly explain how we used Apache Storm, Kafka, and MongoDB using Java codes to create a Realtime Processing system.

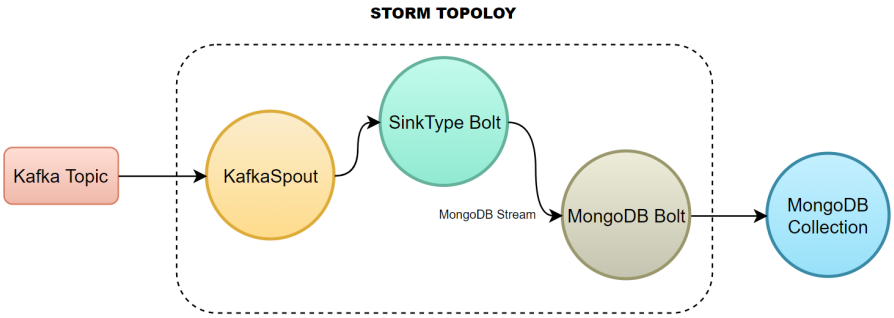
To provide a quick overview of how the system works, we can say messages are sent to a Kafka topic, which is then picked up by Storm via Kafka Spout and passed to a Bolt, parsing and identifying the message type based on the header. Finally, the content of the message is retrieved and transmitted to MongoDB bolt once the message type has been detected.

In this scenario, we will create two Bolts. The first referred to as the SinkType Bolt, functions as a decision-maker. This node is in charge of determining the message type and routing it to the relevant Bolt for persistence.
The main class that connects all of the spouts and bolts is topology. The spout and bolts are linked, as shown in the diagram below. (Figure 3.12)). The Topology class uses SpoutBuilder and BoltBuilder to create all of the spouts and bolts. The TopologyBuilder class connects all of these spouts and bolts. In this scenario, our Storm topology is divided into three bolts:
-KafkaSpout Bolt: Kafka spout reads from the kafka topic.
-SinkType Bolt: This node will operate as the one that is decision-maker. It is able to recognise message types and route the results to the proper Bolt to store.
-Mongodb Bolt: It is used for saving collections in MongoDB.

Firstly, each spout should specify which stream it will get its input from. In this example, Kafka spout reads Kafka topics. So, KafkaSpout must understand how to connect to the



(a) Real-time Processing using Storm-Kafka-MongoDB



(b) Topology for real-time streaming with Kafka, Storm and MongoDB

Figure 3.12 A stream processing Architecture using Kafka-Storm-MongoDB

Kafka broker, the topic's name from which it must read, the zookeeper root, and the consumer group id. The spout uses the zookeeper root and group id to retain the offset information of the till where it has read from the subject. In the event of a failure, the spout can utilize this data to resume reading from the point where it stopped. This example will establish a Kafka spout that will read messages from the incoming topic we built during the Kafka setup.

Moreover, SinkTypeBolt must always listen to the KafkaSpout. Then, It will emit the tuples into the MongoDB stream. SinkTypeBolt makes use of two main methods, declareOutput-Fields and execute. The declareOutputFields method is used to declare which output streams will be emitted by this bolt and what fields will be assigned to each of the tuples that will emit to those streams. In this example, we declare one output stream, and each stream will have two fields, sinkType and content.

The Execute method performs some processing on a Tuple at the moment. The type of the Tuple is determined by the execute method, which reads it. Following that, it calls the collector to send the content to any of the streams. The execute method reads the tuple and determines its type. Afterwards, It calls the collector to send the content to any of the

streams. [57]

Finally, MongoDB Bolt creates a MongoClient instance with a hostname as well as a port, followed by a MongoDatabase instance with the MongoClient and database name. The Tuple input is transformed to an org.bson object. The method getMongoDocForInput returns a document, which is then added to the collection by calling textttMongoDB.getCollection (collection).insertOne(mongoDoc).

3.3 Implementation of Architecture

3.3.1 Installation and Configuration of Servers on cloud

OpenStack cloud computing platform is where we deploy our analytics sandbox. we needed to create a highly reliable infrastructure. So, the best practice was to create cluster with at least three nodes to guarantee high availability, load balancing. A cluster of at least three servers is the best way to create a stable, reliable, and easy-to-manage cluster. [58] Each node requires enough resources to execute all of the combined workloads, as well as some overhead to keep the system up and running and allow for future scalability. This refers to the total amount of CPU, RAM, and storage needed to run everything. To install three instances in cloud infrastructure, we used the dashboard of Compute Canada (Arbutus) [59]. We chose the most compatible image (Ubuntu) from the images lists to work with these big data technologies and Kubernetes. To provide maximum security for the servers, we used SSH keypairs. Another subject we considered during the nodes' creation was adding a security group and managing its rules. For example, we keep open access to the ports that we need to access from outside. In other words, we make it more secure by defining the Ingress rule to a range of dedicated IPs (Remote IP prefix); this gives those users from those specific IP ranges to access to applications through public IP and defines ports. Moreover, every operating system may divide the hard drive into numerous pieces that operate independently by creating disc partitions. We used disk partitioning to keep the data distinct from other operating systems partitions. In this case, If system fails data will remain safe.

3.3.2 Deployment in cluster mode using Kubernetes

In order to familiarize ourselves with architecture, we need to understand kubernetes architecture and main components. Applications in Kubernetes run within containers. A Kubernetes pod is a collection of one or more containers in Kubernetes. Containers in a pod share the same resources and can communicate using localhost or interprocess communication mechanisms. The virtual network of the Kubernetes architecture is used to allow commu-

nication between pods. [28] Kubernetes is using replicsets to ensure that a certain number of pods are running in the system. Kubernetes deployments function as controllers for pods and replicasets, allowing declarative updates to be made. Deployments are a collection of several identical pods with no distinguishing characteristics. When a pod is established in Kubernetes, the available computing resources can be assigned to the containers. Resource requests and limits can be distinguished for the containers of a pod. The minimum value of a specific compute resource that must be guaranteed to the pod is referred to as a request, while the maximum utilization that the pod can utilize is a limit. CPU units and memory were initially supported compute resources. Third-party extensions can be used to provide additional custom resource metrics such as network or storage use. [28] To deploy our analytics sandbox for Real-Time data processing, we need a Kubernetes cluster with a minimum of two workers to provide scalability and two node of masters to enable high availability for the Kubernetes cluster. This configuration for stack guarantees scalability, fault tolerance, and reliability. To address installation and configuration challenges in setting up a multi-node cluster, we created an on-premises Kubernetes cluster using Rancher Kubernetes Engine (RKE) [60] with four nodes; two as a master and the two other as workers. To start building a cluster, we needed some pre-configuration on each node. To benefit from automation, all the required configurations, such as creating a specific user on each node for deployment purposes, enabling related kernel modules, disabling swap, and modifying systemctl entries, and docker installation has been added to Ansible playbooks. Then, we needed to run these Ansible playbooks (https://github.com/ghfaha/BDAH_Sandbox) to prepare all the cluster nodes. The next step was to open firewall ports. Another fundamental configuration was to enable SSH server system-wide TCP forwarding.

Furthermore, each master instance needed to install some CLI tools such as kubectl, RKE, Helm. After installation of necessary CLI tools to work with multi-node Kubernetes, we needed to create a YAML file for cluster configuration. Many configuration options can be set. RKE provides an opportunity to generate default configuration files. The most crucial configuration for cluster YAML files is the role of each node in the Cluster, the address of each node, and network configuration. If all the configuration and pre-installation requirements for each cluster node are correctly configured, once you run the "rke up" command, the Cluster will be running. The Cluster can be managed with kubectl commands.

Each container in Kubernetes can read and write to its isolated filesystem. However, any data on that filesystem will be lost when the container restart. Kubernetes has volumes to help with this. Volumes allow pods to write to a filesystem that exists for as long as the pod does. Users can also use volumes to share data between containers in the same pod. However, data in that volume will be lost when the pod restart. Kubernetes has persistent

volumes to address this issue. In the Kubernetes cluster, persistent volumes are long-term storage. Beyond containers, pods, and nodes, persistent volumes exist. A pod uses a persistent volume claim to gain read and write access to a persistent volume. Firstly, we used a Local Path Provisioner to resolve the problem that the container environment is not persistent by default and create Persistent storage to store data. The Local Path Provisioner enables Kubernetes users to use local storage in each node. The Local Path Provisioner will automatically build a host Path-based persistent volume on the node based on the user's setup. It takes advantage of Kubernetes' Local Persistent Volume capability; however, it is much easier to use it than Kubernetes' built-in local volume feature. It makes use of the functionalities introduced by Kubernetes' Local Persistent Volume feature, but it is a more straightforward solution than Kubernetes' built-in local volume feature. However, the Local Volume Provisioner currently does not support dynamic provisioning for local volumes. NFS Server Provisioner is a solution that dynamically provides Kubernetes back-end storage volumes. It is easy to set up shared storage that works practically anywhere. As a result, NFS-provisioner is another dynamic provisioner solution for Kubernetes. It is essential to consider any data stored on the dynamic volumes provisioned will not be persistent. We used the NFS-provisioner solution for our applications. [61]

3.3.3 Management

Different factors can be considered when it comes to sandbox management, such as resource management and monitoring and etc. Our BDAH platform can support multiple Sandboxes. This is the main reason why we need management. We need to ensure that each sandbox gets the resources it needs and not in the expense of the others. Along with management on a per-sandbox basis, We also need management for the entire infrastructure. Different factors can be considered when it comes to sandbox management, such as resource management and monitoring of tasks. In our Sandbox, Kubernetes is responsible to manage the resources for whole system.

For infrastructure management, there are tools available that are capable to work with kubernetes based systems. Real-time monitoring system is one of them. Real-time monitoring assists sandbox admin in accurately identifying the times an incident occurs, the reporting time, and the resolution time. As a result, health sectors can become more proactive with their response methods and deal with recurring problems more efficiently by identifying these times. In the case of Sandbox, the system's admin can benefit from such a management system to better estimate the amount of resource required to give to a cluster based on the fluctuation in analytics workloads. In addition, this aids them in the auto-scaling

configuration of the system. There are mature monitoring tools available such as NetData, Prometheus. In addition, Grafana is a free and open-source lightweight dashboard application running on Kubernetes. It can be integrated with various data sources, including Prometheus, AWS CloudWatch, Stackdriver, etc.

3.3.4 Auto-scaling

Kubernetes is built to be scalable. It includes a set of tools for scaling a cluster up and down in response to demand based on various other factors. [30] Kubernetes is also capable of automating many management tasks, including provisioning and scaling. It will automate processes that respond quickly to peaks in demand and conserve costs by scaling down when resources are not needed. It can be used in conjunction with the cluster auto-scaler to allocate only the resources required. The auto-scaling mechanism [62] in Kubernetes is made up of two layers:

1. **Pod-based scaling**— supported by the Horizontal Pod Autoscaler (HPA) and the newer version of autoscaler named Vertical Pod Autoscaler (VPA).
2. **Node-based scaling**— supported by the Cluster Autoscaler(CA).

HPA and VPA are two autoscalers that work on the application abstraction layer. Moreover, the infrastructure layer is where CA works.

3.3.5 Kubernetes Scaling Mechanisms

1. Horizontal Pod Autoscaler (HPA):

A mechanism to add or remove pod replicas is needed when the level of application utilization varies. The Horizontal Pod Autoscaler facilitates workload scaling automatically once configured. HPA can be used for stateless applications as well as stateful workloads. HPA is a control loop that the Kubernetes controller manager maintains. The HPA loop's duration is specified by a flag in the controller manager, which is set to 15 seconds by default.

The controller manager compares actual resource use to the metrics established for each HPA at the end of each loop period. If specified, it gets these through the custom metrics API or the resource metrics API that auto-scaling should be dependent on resources per pod (such as CPU consumption).

Scaling is based on a single metric collected from the object, compared to the desired value to give a utilization ratio. [62] Figure 3.13 shows HPA processes.

HPA uses the following metrics to calculate autoscaling:

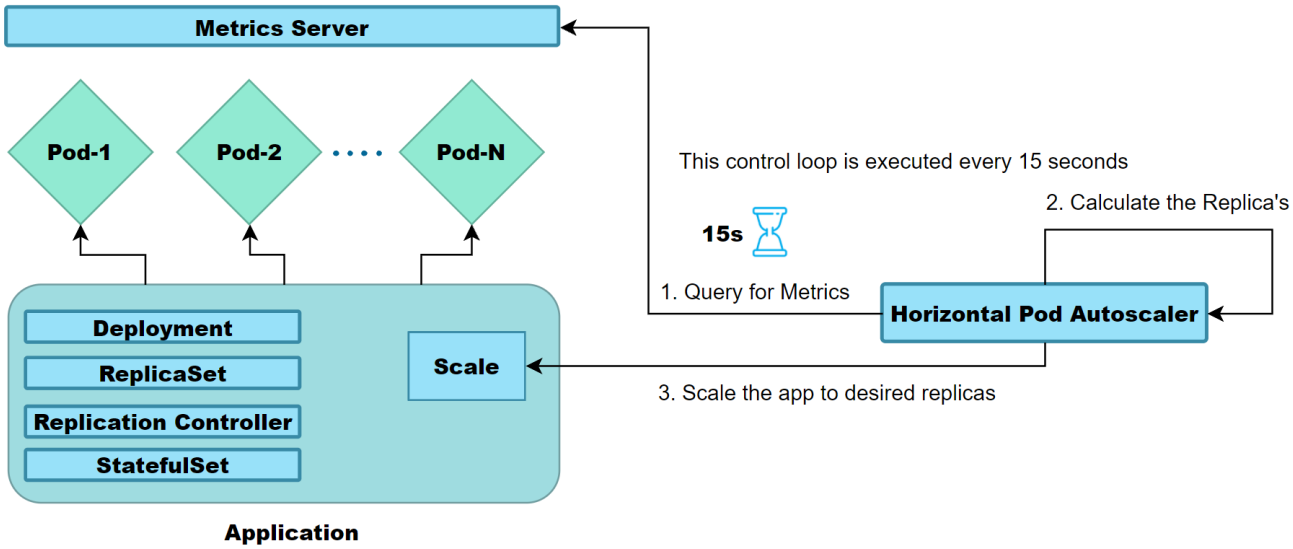


Figure 3.13 Horizontal Pod Autoscaler (HPA)

- **In terms of resource metrics:** A target utilization number or either a fixed target for resource measurements can be set for resource metrics.
- **In terms of custom metrics:** Only raw numbers are supported for custom metrics, and a target utilization can not be set.
- **In terms of object metrics and external metrics:** Scaling is based on a single metric collected from the object, which is compared to the desired value to give a utilization ratio.

Limitation: There is also limitation for Horizontal Pod Autoscaler (HPA). while examining CPU or memory metrics, HPA can not be utilized with vertical pod autoscaling (VPA). Furthermore, when using a Deployment, HPA cannot be configured on a ReplicaSet or Replication Controller, only on the Deployment itself.

Two major best practices for utilizing HPA effectively are as follows: [62]

- Ensure that all pods have resource requests configured— HPA leverages CPU utilization of pods running as a component of a Kubernetes controller when scaling decisions. This is expressed as a proportion of individual pod resource requests. Use all resource request values from all containers to confirm that the data is accurate.

- When possible, utilize custom metrics instead of external metrics; the external metrics API poses a security concern because it allows access to a vast number of metrics. Because it only holds users specific metrics, a bespoke metrics API poses less danger if compromised.
- Use HPA in conjunction with Cluster Autoscaler to coordinate pod scalability with the behavior of nodes in the cluster. When an application needs to scale up, for example, the Cluster Autoscaler can add eligible nodes, and when the application needs to scale down, it can turn off superfluous nodes to save resources.

2. Vertical Pod Autoscaling (VPA):

The Vertical Pod Autoscaler sets resource restrictions for containers based on real-time data. The majority of containers stick to their initial demands rather than their upper limit requests. Kubernetes' default scheduler, as a result, overcommits a node's memory and CPU reservations. To deal with this, the VPA adjusts the number of requests made by pod containers to match the amount of memory and CPU resources available.

Some workloads may necessitate brief bursts of high consumption. By default, increasing request limitations would waste resources and limit the number of nodes available to handle those workloads. In some cases, HPA may assist, but in others, the application may not enable load distribution over numerous instances. The recommender component of a VPA setup calculates target values by monitoring resource usage. Its updater component evicts pods that require resource limitations to be updated. Finally, using a mutating admission webhook, the VPA admission controller overwrites the pod resource requests when they are made. [62]

Limitation: There is also limitation for the Vertical Pod Autoscaling (VPA). Updating operating pods in VPA is still in its early stages, and performance in large clusters has yet to be determined. Second, VPA handles most out-of-memory events, but not all, and the behavior of numerous VPA resources assigned to the same pod is unknown. Finally, while changing pod resources, VPA recreates pods, maybe on a different node. As a result, all containers that are now executing are restarted.

Here are two practices for using Vertical Pod Autoscaler effectively: [62]

- HPA and VPA are incompatible. So, avoid using them in tandem—, so do not use them together. Unless the HPA is configured to use either custom or external metrics, do not use both for the same series of pods.

- VPA should be utilized in conjunction with Cluster Autoscaler because VPA may occasionally recommend resource request levels over available resources. This can put a strain on resources, causing pods to go into a pending state. In response to awaiting pods, the cluster autoscaler can alleviate this behavior by spinning up new nodes.

3. Cluster Autoscaler (CA): HPA scales the number of operational cluster pods, while the cluster autoscaler modifies the number of cluster nodes. Cluster autoscaler looks for unscheduled pods and attempts to condense pods that are only on a few nodes. It is constantly looping over these two activities.

Unschedulable pods are caused by a lack of memory or CPU resources, or by the pod's taint tolerations (rules that restrict a pod from scheduling on a given node), affinity rules (rules that encourage a pod to schedule on a specific node), or nodeSelector labels. If a cluster has unschedulable pods, the autoscaler looks through managed node pools to see if adding a node might help unblock the pod. It adds a node to the pool if the node pool may be expanded.

The autoscaler also monitors the nodes of a managed pool for possible pod rescheduling on other cluster nodes. If any are found, it evicts them and deletes the node. When shifting pods, the autoscaler considers pod priority as well as PodDisruptionBudgets.

The cluster autoscaler offers a 10-minute graceful termination period before forcing a node termination while scaling down. This gives applications enough time to move the node's pods to another node.

The cluster autoscaler adjusts the size of a Kubernetes cluster by adding or removing nodes due to the presence of pending pods and node utilization metrics.

When the utilization of a node drops under a particular limit set by the cluster manager, it is removed from the cluster. The cluster autoscaler is an outstanding tool to enable the elasticity and scalability of the underlying cluster when workloads demand change.

However, there is also the limitation of using it. Cluster autoscaler only works with a few managed Kubernetes platforms; if used Kubernetes platform is not one of them, it is necessary to install it. In addition, Local PersistentVolumes are not supported by the Cluster Autoscaler. When using local SSDs, applications cannot scale up a size 0 node group for pods that need ephemeral storage. Figure 3.14 shows CA processes.

Here are two basic strategies for using Cluster Autoscaler effectively: [62]

- Ascertain that the Cluster Autoscaler pod has sufficient resources by specifying a minimum of one CPU for resource requests to the cluster autoscaler pod. It's vital to

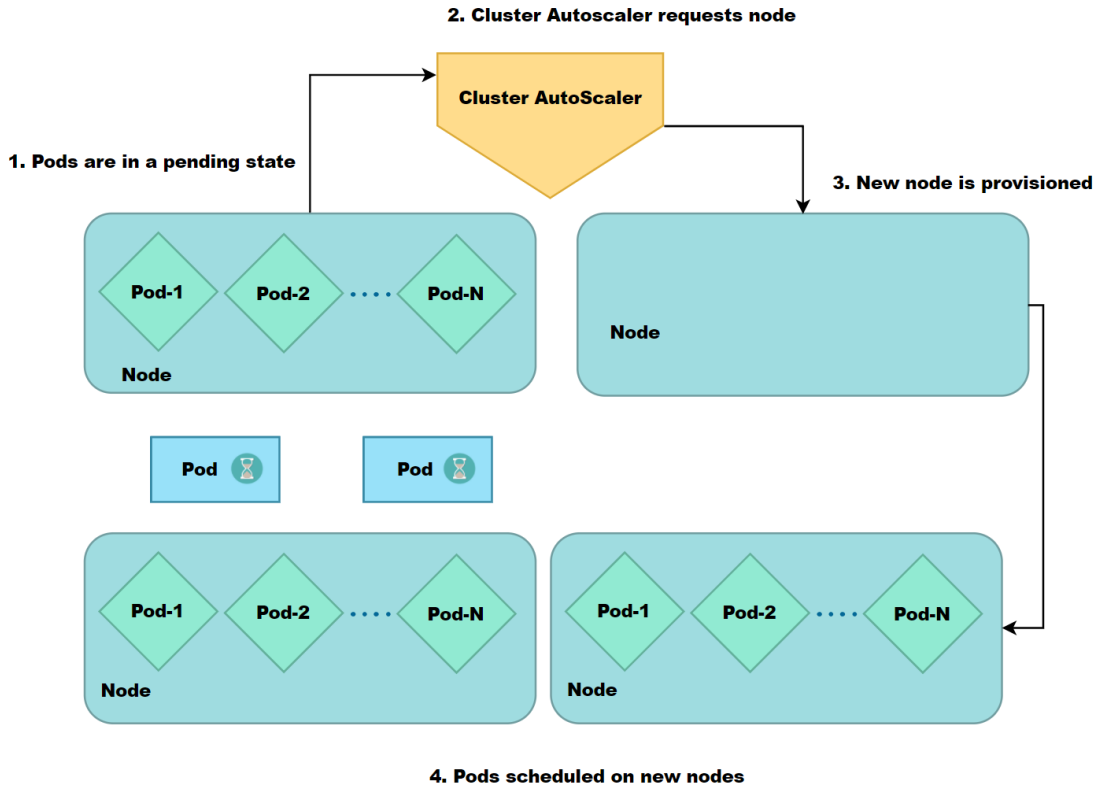


Figure 3.14 Cluster Autoscaler (CA)

make sure the cluster autoscaler pod's running node has enough resources. The cluster autoscaler may become unresponsive if this does not happen.

- Ensure that all pods have a defined resource request—the cluster autoscaler requires a defined resource request to function properly. The reason is the cluster autoscaler takes decisions based on pod status and individual node utilization, and it can be thrown off if the computation is incorrect.

3.3.6 Scaling a Distributed Stream Processor in a Containerized Environment

Kubernetes has established itself as the de-facto platform for clustering containerized workloads. Individual distributed applications and services can be deployed and managed using their native abstractions. On the other hand, streaming applications necessitate the simultaneous deployment of many components such as topics, storage, and application runtimes. Furthermore, the expectations of end-users, mostly data scientists and data engineers, are to have a system up and running 24 hours a day, even if system load is variable in time.

We are aware that Kubernetes will scale up as soon as the system needs to be scaled and we

are aware of the benefits it can bring to data analytics. To satisfy end-user demand, We'd like the number of nodes in a cluster and the number of pods in deployment to adjust dynamically based on the workload. To test the autoscaling of streaming apps in Kubernetes, We used a

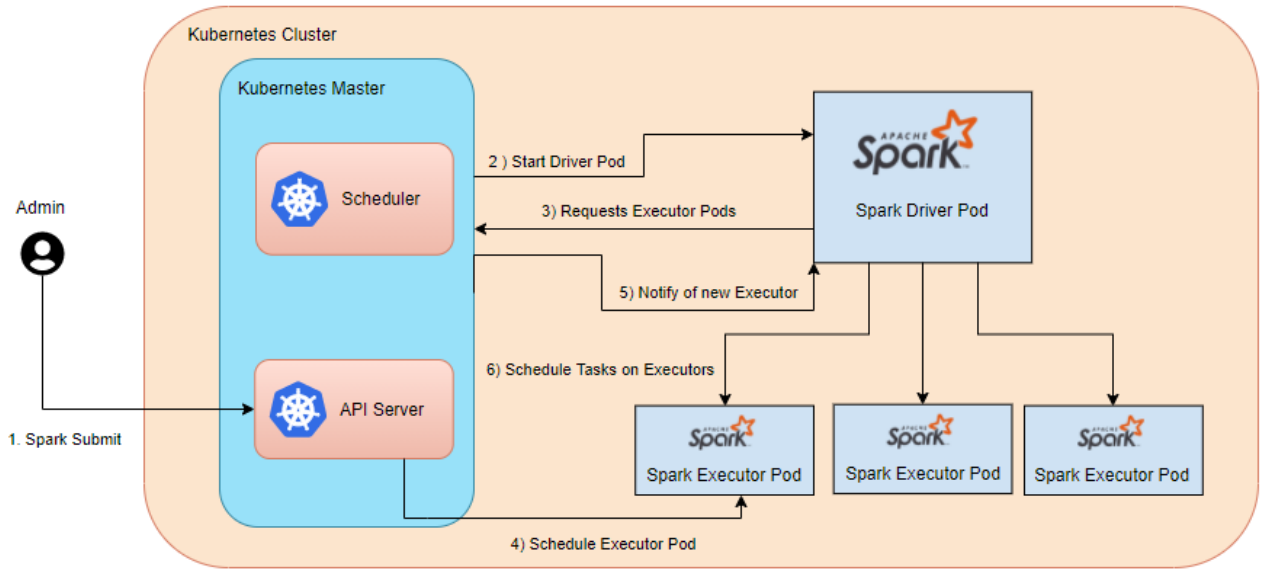


Figure 3.15 Architecture of Spark Cluster in Kubernetes

scenario to put more loads on our app to monitor the behavior of Kubernetes. First, however, it is essential to configure Kubernetes to scale in case of an increase in load. In this example, we are using Horizontal Pod Autoscaler. Before that, we should make sure our deployment is running. Then we will be able to configure autoscaling for each deployment with the number of replicas of the Pods and the maximum CPU percentage to trigger autoscaling.

```
kubectl autoscale deployment spark-123456 -cpu-percent=50 -min=1 -max=10
```

To do so, We used the Hibench streaming benchmark tool to run the spark-terasort workload on spark running in Kubernetes for about 30 min. The number of pods will increase when the system CPU load reaches the amount we determine until we satisfy the spark need for the load. We monitor the whole process using Netdata and kubectl commands until we terminate the loads and the pods drop to the one pod as before. Figure 3.15 illustrates the architecture of Spark cluster in kubernetes.

During the autoscaling process, We inspected the autoscaler's present state by running: `kubectl get hpa`, [28] which shows Horizontal pod autoscaling current status of each deployment on the Kubernetes cluster and metrics such as TARGET (CPU utilization for autoscaling.), MINPODS, MAXPODS, etc. To check the number of pods, we used `kubectl`

`get pods`.

3.4 Multi-Tenancy in Sandbox

When various Analytics teams are communicating with their individual sandbox in our BDAH platform, multi-tenancy must be considered. Most industries consider a multi-tenant platform to operate their cloud-native apps because it permits them to manage better resources, expenses, and operational efficiency while also reducing cloud waste. Users and/or workloads are referred to as "tenants" in a multi-tenant cluster. To minimize the damage that a compromised or hostile tenant can cause to the cluster and other tenants, multi-tenant cluster operators must segregate tenants from one another. Cluster resources must also be distributed evenly among tenants. Consider the layers of resource isolation in Kubernetes when planning a multi-tenant architecture: cluster, namespace, node, Pod, and container. we should also think about the security implications of letting renters share various types of resources. Scheduling pods from various tenants on the same node, for example, might decrease the number of machines required in the cluster. In either case, it is necessary to avoid specific colocating workloads. To prevent an untrusted code from outside the organization to execute on the same node as malicious insider containers, for example. [63]

3.4.1 Tenancy Models For Kubernetes

Operating a multi-tenant cluster has a wealth of advantages over managing several single-tenant clusters such as; decrease in Management overhead, Reduced fragmentation of resources, There is also no need to wait for a cluster to be created before bringing on new tenants. Although Kubernetes cannot guarantee absolute tenant isolation, it does provide characteristics that may be sufficient under usual scenarios. [63]

1. **Namespaces as a Service:** Tenants share a cluster in the namespaces-as-a-service paradigm, and tenant workloads are limited to a set of Namespaces allotted to the tenant. All tenants can use the cluster control plane resources like the API server and scheduler, as well as worker node resources like CPU, memory, and so on. [64] Each namespace must also include the following to isolate tenant workloads:

   (a) **Role bindings:** Role bindings are used to restrict access to the namespace.

   (b) **Network policies:** The goal of network policies is to keep network traffic from passing amongst tenants.

(c) **Resource allocation:** Quotas are used to restrict resource utilisation and ensure that tenants are treated equally.

Tenants can't generate or alter cluster-wide resources like ClusterRoles and CustomResourceDefinitions (CRDs) under this paradigm because they share them.

By allowing users to create additional namespaces under a namespace and propagating resources within the namespace hierarchy, the Hierarchical Namespace Controller (HNC) project makes it easier to manage namespace-based tenancy. This allows tenants to create self-service namespaces without having cluster-wide permissions. The Multi-Tenancy Benchmarks (MTB) project provides benchmarks as well as a command-line utility that conducts many setup and runtime checks to determine whether tenant namespaces are correctly segregated and security restrictions are in place.

2. **Clusters as a Service:** Each tenant receives their own cluster when using the clusters-as-a-service paradigm. This paradigm allows tenants to have distinct versions of cluster-wide resources like CRDs, yet the Kubernetes control plane is fully isolated. Projects like Cluster API (CAPI), which uses a management cluster to provision several workload clusters, can be used to provision tenant clusters. A tenant is assigned to a workload cluster, and tenants have complete authority over cluster resources. It's worth noting that in most companies, a central platform team is in charge of administration essential add-on services like security and monitoring, as well as cluster lifecycle management services like patching and updates. It's possible that a tenant administrator won't be able to change the centrally managed services or other key cluster data. [64]

3. **Control planes as a Service:** The tenant cluster may be a virtual cluster under a variant of the clusters-as-a-service model, where each tenant has their own private Kubernetes control plane but shares worker node resources. Users of a virtual cluster observe no substantial differences between a virtual cluster and other Kubernetes clusters, as they do with other forms of virtualization. Control Planes as a Service is another term for this (CPaaS). This kind of virtual cluster shares worker node resources as well as task state independent control plane components such as the scheduler. To accommodate conflicts, other workload-aware control-plane components, such as the API server, are constructed per-tenant, and additional components are employed to synchronise and maintain state across the per-tenant control plane and the underlying shared cluster resources. Users can control cluster-wide resources using this model. This paradigm is implemented by the Virtual Cluster project, in which a supercluster is shared by several virtual clusters. The Cluster API Nested project is developing

this work to adhere to the CAPI architecture, allowing users to provision and manage virtual clusters using familiar API resources. [64]

Using one or more of these models in our proposed data analytics stack can mainly guarantee the reliability and integrity of analyses in data access. In a nutshell, namespaces as a service tenancy concept promote resource efficiencies by allowing clusters to be shared. However, because all tenants use the same cluster-wide resources, it necessitates suitable security measures and has constraints. The clusters address this constraint of namespaces as a service model as a service tenancy paradigm but at the cost of higher management and resource waste. For instance, in a cluster as a service model, one cluster cannot have access or interfere with data or results from another cluster. Moreover, as another example, in control planes as a service model, the tenant cluster owner will share worker node resources. The control planes as a service paradigm enable tenants to share Kubernetes cluster resources while also managing their cluster-wide resources. Sharing worker node resources improves resource efficiency, raising questions about cross-tenant security and isolation in shared clusters. [64]

CHAPTER 4 EVALUATION

As described in the previous chapter, certain components of the proposed platform can be implemented using various tools. In order to provide better insight and guide the users to choose the best tool for their purposes we present a series of comparative experiments where we compare the performance of certain alternatives (with respect to ML analyses and streaming), so that the users can choose which tool fits best based on their anticipated workloads and resource constraints. In fact, the section includes a comparison between Spark and Storm on streaming analytics as well as a comparison between Spark and Hadoop on ML workloads, using a big data benchmarking suite called HiBench [65].

4.1 Performance Evaluation Processes and Tools Used

To evaluate and characterize the popular analytics tools such as Hadoop, Spark, Storm, and Storm trident in terms of latency (i.e., job running time), we used Hibench.

HiBench is a tool that can compare different big data frameworks in terms of latency, throughput, and system resource usage. It includes various workloads such as Sort, WordCount, TeraSort, Repartition, Sleep, SQL, PageRank, Nutch indexing, Bayes, Kmeans, NWeight, and enhanced DFSIO for Hadoop, Spark, and streaming workloads. [65] However, since the streaming section does not provide a report with system resource usage, we used another tool called Netdata. Netdata is a real-time monitoring system that provides a complete result of various resource usages during each workload process. In addition, the results can be aggregated from all the nodes' resource usage.

The benchmarking of Big Data systems can be split into three key elements or aspects, which are usually carried out in order. The first stage is to generate workload input data. Following the generation of the data, an analytic job is executed with the data as input. Finally, a report is generated that includes several parameters such as latency, execution time, etc. Figure 4.1 shows the process flow diagram for stream-based Benchmarking in HiBench.

The components of the streaming benchmark are as follows:

- **Data Generation:** The data generator intends to produce consistent data and transfer it to the Kafka cluster. Therefore, when a record is created, it is given a timestamp.
- **Kafka cluster:** The data is saved to Kafka topics by each streaming workload.
- **Stream processing tool in cluster mode:** A Spark, Storm cluster could be used.

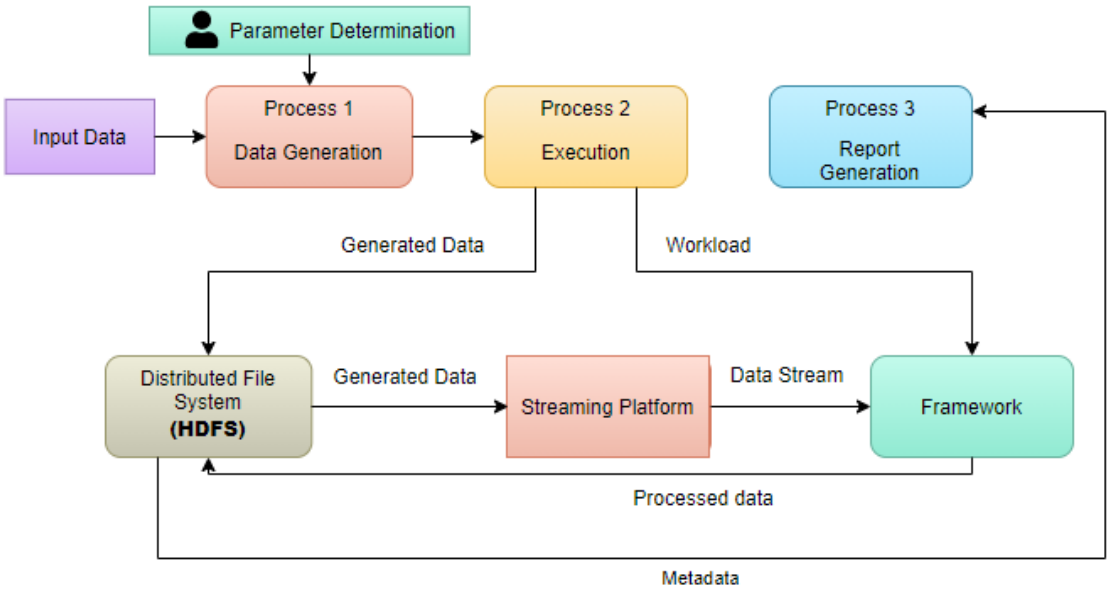


Figure 4.1 Process flow diagram for stream based Benchmarking in HiBench

The streaming application gathers data from Kafka, processes it, and writes the results back to another Kafka topic (Identification, Repartition, Wordcount, Fixwindow). A timestamp is also assigned to each entry in the result.

- **Metrics reader:** It reads the result from Kafka and calculates the time difference (Record out time - Record in time) before producing the report.

Software Requirement	
OpenStack Flavor	Centos 7
Java JDK	V1.8
Maven	3.8.3
Python	V2.7
Hadoop(cluster Mode)	V3.2.2
Hibench	V8
Kafka(cluster Mode)	V0.8.2.2
Zookeeper(cluster Mode)	V3.4.8
Spark (cluster Mode)	V2.2.0 (Streaming) V3.0.3(ML)
Storm (Cluster Mode)	1.0.1

Hardware Configuration	
Openstack	
Instance Type	Nova
Instance Numbers	Three
Processor(VCPUs)	40
Memory	3*90GB
Disk	1.5TB

Table 4.1 Software and Hardware Specifications

4.2 Hardware and software specification

The experiments were carried out on a Compute Canada OpenStack cluster. The cluster is configured with one master and two slave nodes. As shown in the Table 4.1, the cluster is equipped with a total of 40 VCPU cores and 1.5 TB of local storage across all nodes. The hardware is capable of dealing with a variety of challenging situations in Spark and MapReduce and Storm. We have used Netdata to monitor the selected workloads resource usage in all the nodes and also in aggregation amount.

HiBench Advanced Configuration	
hibench.scale.profile	Data scale profile `tiny`, `small`, `large`, `huge`, `gigantic`, `bigdata`
hibench.default.map.parallelism	Mapper numbers in Map Reduce/Partition numbers in Spark
hibench.default.shuffle.parallelism	Reducer numbers in MRshuffle/ Partition numbers in Spark
hibench.yarn.executors.num	Number executors in YARN mode
hibench.yarn.executors.cores	Number executor cores in YARN mode
spark.executors.memory	Executor memory, standalone or YARN mode
spark.driver.memory	Driver memory, standalone or YARN mode
hibench.compress.profile	Compression option `enable` or `disable`
hibench.compress.codec.profile	Compression codec, `snappy`, `lzo` or `default`

Table 4.2 HiBench Advanced Configuration

4.3 Tuning approach and parameters of Interests

There are advanced configuration options in HiBench that it is necessary to familiarize oursleves with them before start benchmarking tools. These parameters have been shown in Table 4.2. [66] Furthermore, it is also critical to tune parameters for big data computing applications such as Hadoop, Spark and Storm before benchmarking. We need to understand which parameters have a significant impact on system performance. The parameter configuration must be investigated concerning workload, data size, and cluster architecture.

Framework	Related Configuration
Hadoop	• mapreduce.map.memory = 6 GB • mapreduce.reduce.memory = 8 GB • mapreduce.reduce.cpu.vcores = 4 • mapper.number = 12 • reducer.number = 8
Spark	• spark.driver.memory = 6 GB • spark.executor.memory = 8 GB • spark.executors.num = 2 • spark.executor.cores = 4 • spark.default.parallelism = 12 • spark.sql.shuffle.partitions = 8

Table 4.3 Spark and Hadoop Configuration

configuration	Large	Huge	Bigdata
num of clusters	5	5	5
dimensions	20	20	20
num of samples	20000000	100000000	1200000000
samples per inputfile	4000000	20000000	4000000000
max iteration	5	5	10
k	10	10	10

Table 4.4 K-means Tuning Parameters

These parameters are investigated for each experiment.

- Batch-ML Experiment: We ran some experiments with Apache Hadoop and Apache Spark, each with a different set of parameters. We chose the core MapReduce and Spark parameter settings from resource utilization and shuffle groups for this experiment. Table 4.3 displays related configuration on the map-reduce and Spark categories, along with their adjusted values. Furthermore, Table 4.4 illustrates the tuned parameters for each profile.

- Streaming Experiment: Before we begin to do benchmarking we required to do some configuration in Hibench to make sure there is no bias within the results between the results of spark and storm. Table 4.5 displays the tuned parameters on Storm and Spark.

Framework	Related Configuration
Spark Streaming	• Executor Number = 2 • Map Parallesim =12
Storm	• Worker = 8 • KafkaSpout= 12

Table 4.5 Streaming Frameworks Benchmarking Related parameters Configuration

4.4 Hadoop vs Spark

4.4.1 Functional comparison of Hadoop and Spark

Hadoop is a framework that employs the MapReduce (MR) model. In contrast, Spark is a technology for fast cluster computation that amplify the MR model to handle a broader

range of computations efficiently. [39]

In terms of performance,Hadoop's MapReduce model reads and writes to disk, slowing processing performance, whilst Spark decreases disk read/write cycles and stores intermediate data in memory, resulting in quicker computing speed. [67]

From perspective of ease of use, Hadoop requires developers to program each operation, whereas Spark uses RDD to make programming easier.

Because the Hadoop MR model only includes a batch engine, it relies on other engines for other necessities. On the other hand, Spark can run batch processing, interactive queries, ML, and stream processing in the same cluster. [67]

In terms of performance, Hadoop is a high-latency computing framework that lacks an interactive mode. whilst, Spark is a processing framework that can process the data interactively and with low latency. [67]

Hadoop is built to manage faults and failures. The fact that it is resilient to faults, making it an excellent fault-tolerant system, although RDD in Spark allows partition recovery on failed nodes. [67]

To schedule complex flows, Hadoop requires an external job scheduler, such as Oozie, whereas Spark uses in-memory computation and thus has its own flow scheduler.

In terms of cost efficiency, Hadoop is the less expensive option in terms of cost-efficiency, whereas Spark needs a lot of memory to run tasks, increasing the cluster and thus the cost. [67]

From the perspective of scalability, as data volume multiplies, Hadoop scales rapidly to meet the demand through the HDFS. In turn, for large amounts of data, Spark depends on the fault-tolerant HDFS. [39]

In terms of security, Spark enhances security by using a shared secret or event logging, while Hadoop employs a variety of authentication and access control techniques. Thus, although Hadoop is more secure, Spark can incorporate it with Hadoop to achieve a satisfactory degree of security. [39]

Because it includes MLlib, Spark is the outstanding platform in this category for better machine learning support capability. Spark MLib performs iterative in-memory ML computations. It also contains regression, classification, persistence, pipeline construction, and evaluation techniques, among other things. [39]

4.4.2 Batch-ML Experiment

In this section, we evaluate the K-means clustering algorithm performance using Hibench for HadoopBench and SparkBench. This workload implements K-Means, which is a prominent clustering algorithm for data mining and knowledge discovery. This benchmark accepts a sample set as input, with each sample presented as a numerical D-dimensional vector. [68] The centroid of each cluster is computed first in this workload by iteratively running the Hadoop task until the number of iterations reaches the set limit. The clustering process is then executed, which assigns each sample to one of the clusters. The workload input is generated using a random data generator with a statistic distribution. For this benchmark, GenKMeansDataset generates the input dataset based on uniform and Gaussian distributions. Zhan et al. [69] mentioned that throughout iteration, this process is CPU-bound, and during clustering, it is I/O-bound.

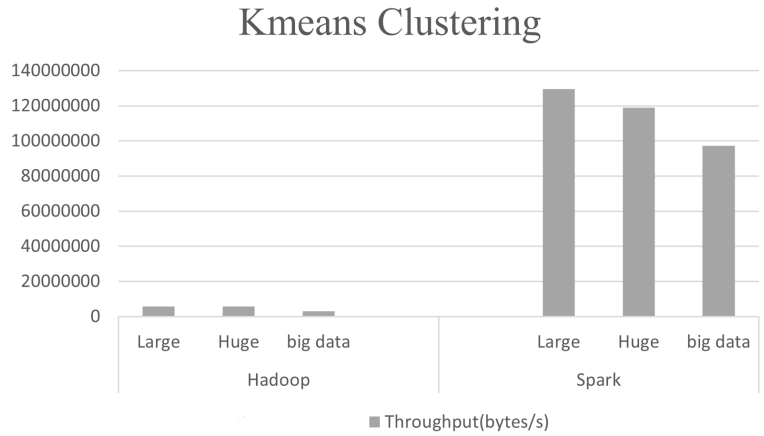


Figure 4.2 Hadoop and Spark Kmeans-Clustering Workload throughput

The goal of running this workload is to compare K-means results between Hadoop and spark in terms of throughput. We tried to compare the results by changing the three Hibench

	Hibench.profile	**Duration(s)**
Hadoop	Large	702.089
	Huge	697.906
	Big Data	1256.161
Spark	Large	30.703
	Huge	33.435
	Big Data	40.985

Table 4.6 Hadoop and Spark Batch-ML Duration

profiles, each with distinct parameters values. Table 4.5 illustrates the parameters values in each performed profile. The results of benchmarking are shown in figure 4.2. The results show spark outperforms Hadoop during all testing profiles. Table 4.6 illustrates the duration of each HiBench profile for Hadoop and Spark.

4.5 Stream Analytics Storm vs Spark

4.5.1 Functional comparison of Storm, Spark and Trident

Storm supports actual stream processing by the core layer in its process model, while Spark streaming acts mainly as a wrapper for Spark batch processing. From the perspective of message delivery, Storm has a processing mode called "exactly once." It can also be used in "at least once" and "at most once" processing modes. However, The "exact once" processing mode is the mode that is enabled by Spark streaming. Trident is also use "exact once" processing mode. [70]

Ackers[1] in Storm are aware of whether or not a record has been successfully processed. If it does not succeed, try afterward. In Storm, there is no guarantee of state consistency. In the event of a failure, Storm uses the acking mechanism to replay Tuples. The state may be committed, but the worker may crash before acking the Tuples. The Tuples are replayed in this situation, resulting in repeated state update. In addition, Checkpointing with the streaming context is required in stateful streaming so that the state can be restored if necessary. In Spark Streaming, checkpointing is used to provide fault tolerance. Essentially, the intermediate values are saved in a storage system, preferably one that is fault-tolerant, such as HDFS. As a result, state in spark is persisted in durable storage since Checkpoint is tied with state storage per Batch. In Storm core, Bolts can save and retrieve the state of their operations using abstractions. There is a default in-memory-based state implementation as well as a Redis-backed state persistence approach. [71] In Spark streaming, the updateStateByKey API and in newer versions mapWithState API can be used to maintain and change state. In Storm, KeyValueState uses. In addition, in Storm Trident, state constancy is guaranteed using persistentAggregate. Figure 4.3 illustrates fault-tolerant mechanism in Storm, Spark and Trident. This is very important part since it affects lots of features.

Storm has a large number of primitives for performing tuple level processing at stream intervals. Aggregations of messages in a stream are achievable using group by semantics. Storm supports right join, left join, and inner join (default) throughout the stream. Besides, There are different sorts of streaming operators in Spark streaming: stream transformation opera-

[1] Storm's acker tracks completion of each tupletree with a checksum hash [45]

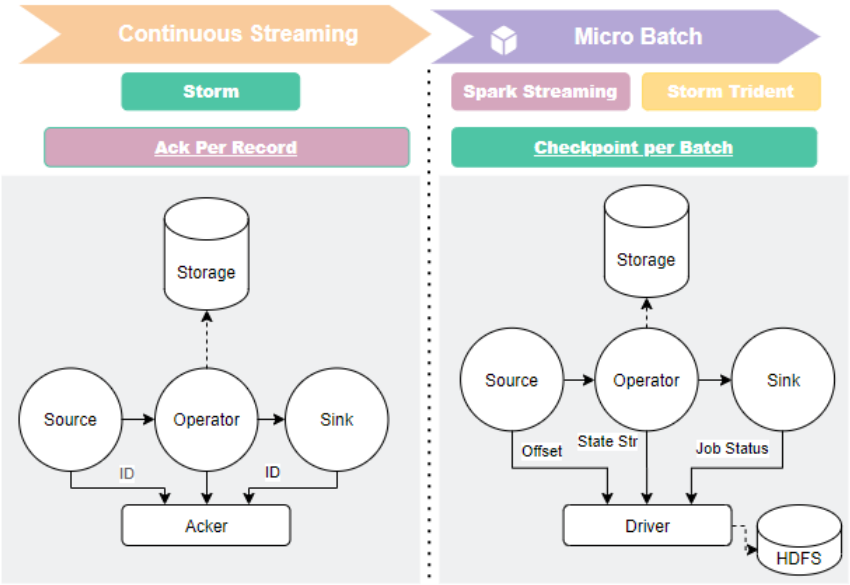


Figure 4.3 Spark and Storm Fault-Tolerant Mechanism

tors and output operators. When we discuss over stream transformation operators, it means to transfer one DStream into another. Output operators are those who write data to external systems. [70]

Storm is built with fault tolerance as its core. If the process fails, the supervisor process will immediately restart it. As ZooKeeper is in charge of managing the state. Spark Streaming is also built to be fault-tolerant. Spark uses resource managers like Yarn, Mesos, or its Standalone Manager to restart workers.

From an operability standpoint, it is not easy to deploy/install Storm through numerous tools and launches the cluster. Storm is controlled by the Zookeeper cluster, which handles cluster coordination and stores cluster state and statistics. Storm daemons are also forced to run in supervised mode rather than standalone mode. On the other other hand, Spark can also manage by an application master in YARN mode, feeding a YARN spark cluster is not difficult. However, in the sandbox, resource management is handled by Kubernetes instead of YARN. [70]

Apart from that, every topology is accessible through Storm UI. This aids in high-level problem debugging and enables metric-based monitoring. The built-in metrics feature helps programs to emit any metrics at the framework level. Furthermore, external metrics/monitoring systems can be easily incorporated with this. In addition, spark web UI displays statistics of running receivers and accomplished tasks. In addition, observing the application's execution

information is also beneficial. Also, the following information in the Spark web UI is required for batch size standardization:

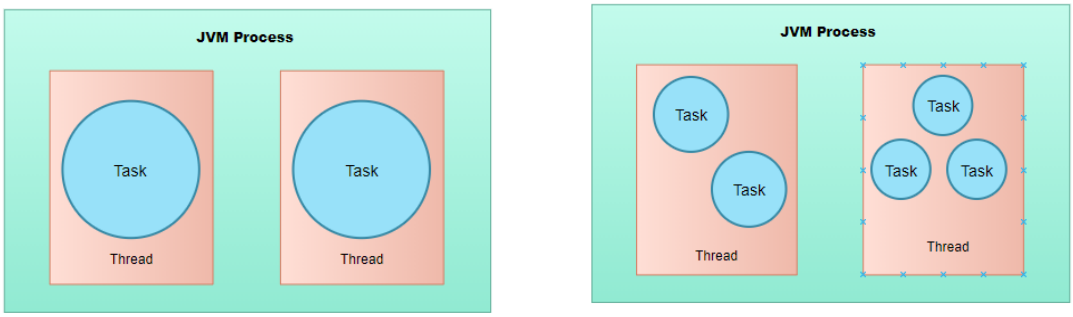
Processing Time - This is the amount of time it takes to process each batch of data. **Scheduling Delay** - This is the amount of time a batch sits in a queue waiting for preceding batches to finish processing. [70]

Each worker process in Storm runs executors for a specific topology. At the worker process level, mixing multiple topological jobs is not permitted. Despite this, topology-level runtime isolation is supported. In addition, the Spark executor runs in a separate YARN container in Spark streaming. As a result, Yarn provides JVM isolation. Because two distinct topologies cannot run in the same JVM; Instead, YARN enables resource segregation at the container level, allowing container restrictions to be arranged. [70]

Compositional and Declarative programming models are two types of programming models. The compositional approach provides essential building parts such as sources and operators, which must be connected to form the desired topology. In most cases, new components are defined by implementing some form of interface. Operators in declarative API, on the other hand, are defined as higher-level functions. It enables developers to write functional code with abstract types and associated gizmos while the system automatically generates and optimizes topology. Declarative APIs are also more likely to include complex features like windowing and state management out of the box. Compositional API offers fully configurable operators based on fundamental building blocks. It allows us to define and optimize topologies manually. Storm can be categorized into this type. In the Declarative API, higher-order functions (map, filter, mapWithState...) are available as operators and logical plan optimization. Spark Streaming and Trident are categorized in this type. [71]
Python and R are two common dynamically-typed languages used mainly by data scientists, supported by spark streaming and Storm but not in Trident. [71]

There are four different run time models listed as below: [71]

1. Single Task on Single Process
2. Multi Tasks of Multi Applications on Single Process
3. Multi Tasks of Single application on Single Process:
    - Single task on single thread which supported by Spark Streaming.
    - Multi tasks on single thread which supported by Storm and Trident.



(a) Single task on single thread (b) Multi tasks on single thread

Figure 4.4 Run Time Model of Storm and Spark

Figure 4.4 illustrates the run time models of execution for Spark and Storm. Table 4.7 summarizes spark and storm and trident framework characteristics. [72]

Table 4.7 Overall comparison of Spark, Storm and Trident

Framework Characteristics	**Streaming Tools**		
	Storm	**Spark**	**Trident**
Assurance	At Least Once/ At Most Once	Exactly Once	Exactly Once
State-fullness	Yes	Yes	Yes
Flow of Data	DAG	DAG	DAG
Community	Selective	Wide	Selective
Streaming Type	Native Streaming	Micro-Batching	Micro-Batching
API	Compositional	Declarative	Declarative
Scaling	Manual	Auto	Manual
Language	Java, Clojure, Scala	Scala, Java, Python	Java, Clojure, Scala
Data Carrier	Tuple Stream	Dstream	Tuple Batch
Maturity	High	High	High

4.5.2 Streaming Experiments

HiBench has a total of 29 workloads. Micro, ml (machine learning), sql, graph, websearch, and streaming workloads. Among all, there are workloads specified only for Streaming Benchmarks purposes which are classified into 4 groups: [65]

1. Identity (identity):
This workload goal is to read data from Kafka and rapidly start writing the output to Kafka;

there is no complex business logic involved.

2. Repartition (streaming/repartition):

This workload gets data from Kafka and adjusts parallelism by dividing the data into more or fewer partitions. It evaluates the effectiveness of data shuffle in streaming systems.

3. Stateful Wordcount (wordcount):

Every several seconds, this workload calculates the number of words received from Kafka. This benchmarks the performance of stateful operators and the cost of Checkpointing and Acking mechanisms in streaming platforms.

4. Fixwindow (fixwindow):

A window-based aggregation is performed by the workloads. It evaluates the performance of streaming frameworks' window operations.

Benchmarking related parameters configuration for streaming frameworks has been shown in Table 4.5. This study has been done to compare and assess the performance and efficiency of big data tools using the HiBench benchmark suite. This section only concentrates on Stream processing tools benchmarking for streaming applications like Apache Spark, Storm, Trident. We run all of the aforementioned workloads on the Spark, Storm, Trident five times each for 30 minutes. Latency results are plotted for all streaming workloads. Figure 4.5 illustrates all the workloads results. Based on the results on Streaming Workload Results of boxplotR figures, the latency of all types of workload for Storm was the lowest. Although, results of latency for identity and wordcount workloads were more closer to spark.

To measure resource usage, we run Identity workload (with bigdata profile) on each framework and for 30 minutes. After that, we used Netdata to monitor the sum of all three nodes' resource utilization. In addition, to ensure there is no bias in results, we wait about 2 hours between every test. Furthermore, we used this command to remove all system memory caches and swap in between the tests.

```
$ su -c "echo 3 >'/proc/sys/vm/drop_caches' && swapoff -a && swapon -a &&
printf '\n%s\n' 'Ram-cache and Swap Cleared'" root
```

In terms of memory consumption, Spark consumes the most. However, Storm was also was high memory consumer during its operation. However, it uses mainly cache memory (55.0 GB) and less used memory (20.7GB) compared to Spark. Section (b) of figure 4.6, 4.7 and 4.8 illustrates the consumption of memory respectively in Spark, Storm and Trident.

In terms of CPU usage, Storm was the one that consumes the most CPU during workload running. (around 50% of the overall cluster's CPU amount was busy during its operation) In comparison, CPU usage in Storm Trident was the least, which makes it ideal for CPU-

intensive stream computing tasks. Section (a) of figure 4.6, 4.7 and 4.8 shows the consumption of CPU respectively in Spark, Storm and Trident.

Spark puts more load and stress on the system; however, this amount is less for Storm and a lot less for Trident. Section (c) of figure 4.6, 4.7 and 4.8 illustrates the system load in Spark, Storm and Trident.

In terms of disk I/O, Spark has only read operation from disk. Storm has more write operation than read. Trident also only read from disk. Section (d) of figure 4.6, 4.7 and 4.8 depicts disk I/O consumption respectively in Spark, Storm and Trident.

In terms of Network usage, Spark has the maximum network consumption while its processes. Section (e) of figure 4.6, 4.7 and 4.8 illustrates the consumption of network respectively in Spark, Storm and Trident.

Since Repartition workload can represent the efficiency of data shuffle in streaming frameworks, results can prove the best ability of data shuffling in Spark.

Whereas Wordcount workload measure the performance of stateful operators, the low latency of Storm between Spark and Storm proves that the cost of Checkpointing is more than Acking fault-tolerance mechanism.

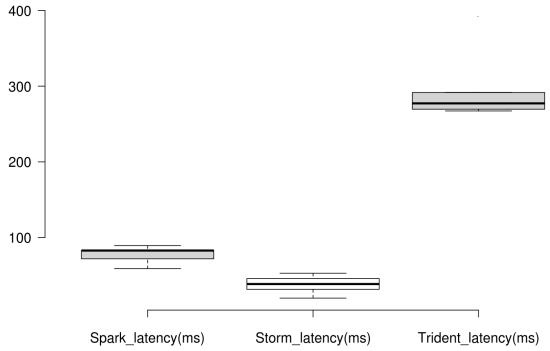
According to latency results for Fixwindow workload, we can conclude windowing in Storm is faster than other frameworks.

In summary, Trident has the lowest resource usage between the three streaming tool such as CPU, memory, network, etc. Furthermore, it has the highest latency between the other tools in streaming workloads.

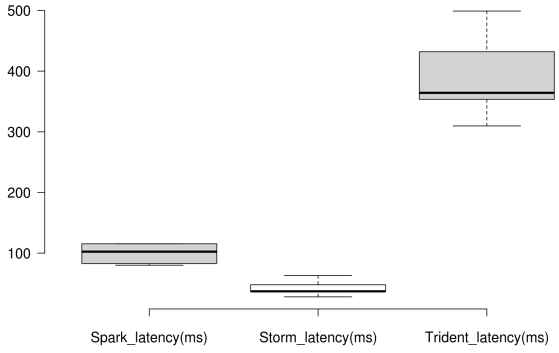
We can also summarize the results as a guide for the users listed as below:

- Spark has the best ability of data shuffling
- Cost of Checkpointing is more than Acking fault-tolerance mechanism.
- Windowing in Storm is faster than Spark
- Storm Trident is the ideal tool for CPU intensive stream computing tasks.
- Spark uses mostly cache memory while Storm uses memory during workload execution.
- The best tool in terms of memory usage is Storm trident.
- Spark has more write operation than read while Storm has mostly read operation while processing data

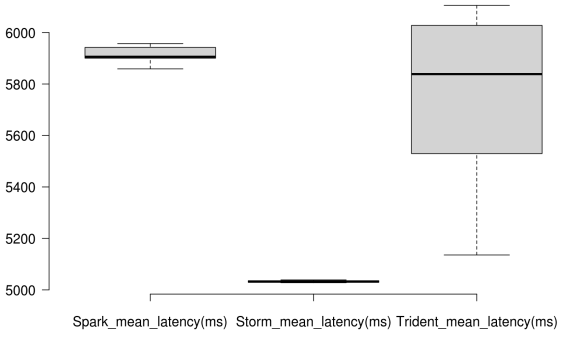
- Spark uses the most network usage
- Storm and Storm trident put the least load and pressure on the system.



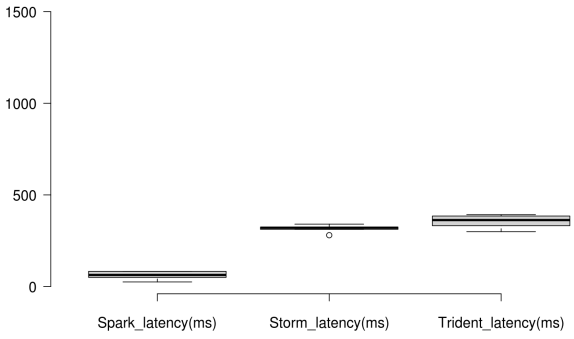
(a) Identity Workload



(b) Wordcount Workload

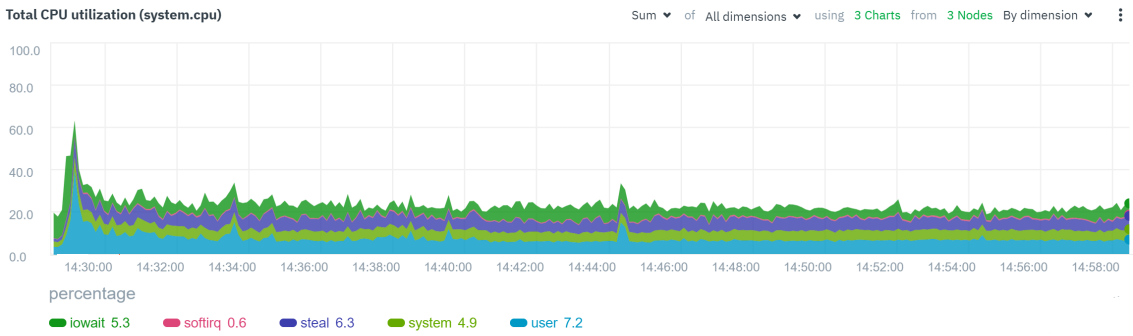


(c) Fixwindow Workload

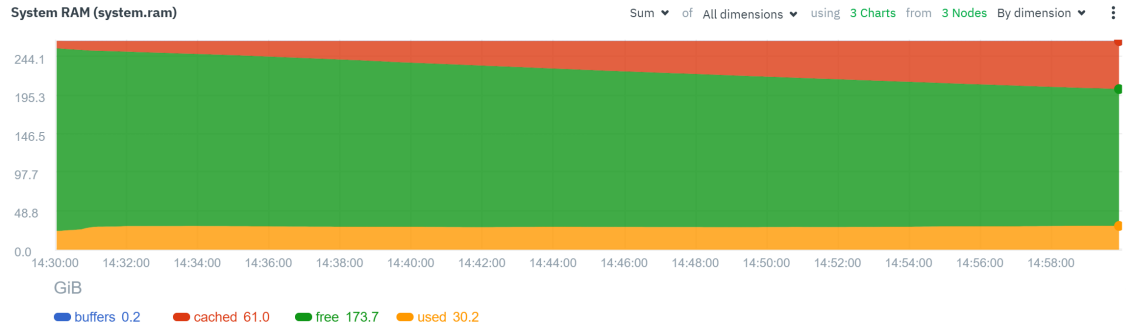


(d) Repartition Workload

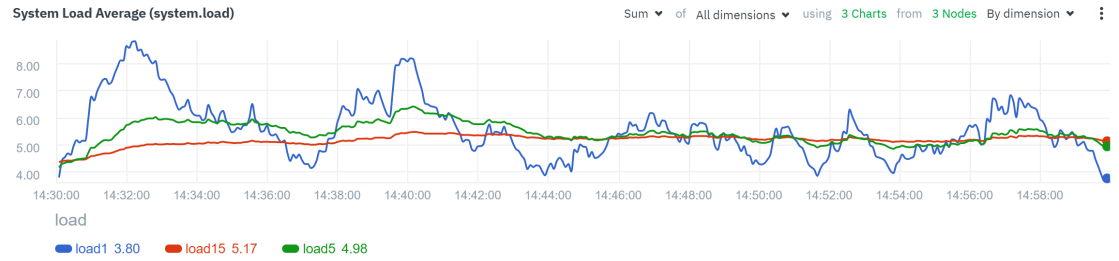
Figure 4.5 Streaming Workloads Latency Results for Spark, Storm, Trident



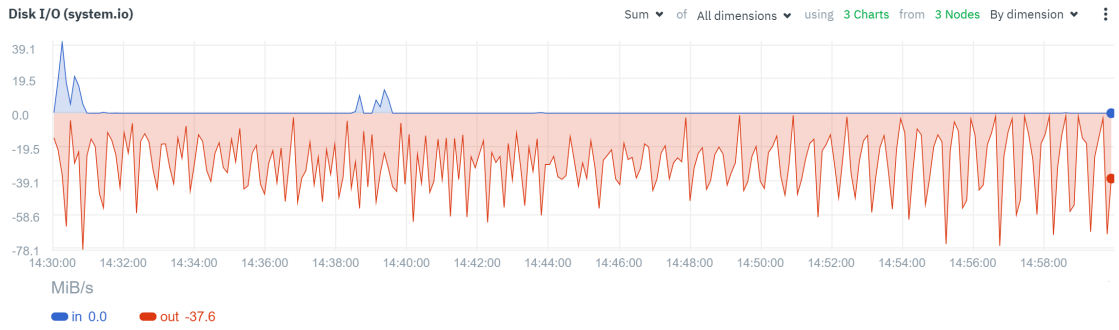
(a) Spark CPU Utilization



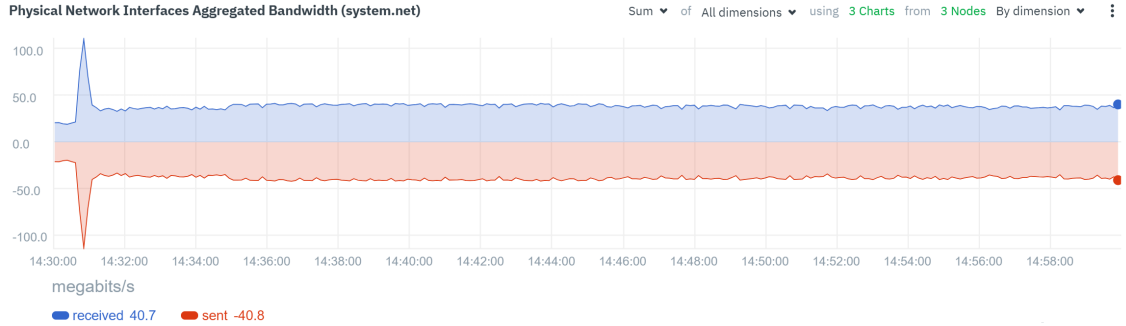
(b) Spark Memory Usage



(c) Spark System Load

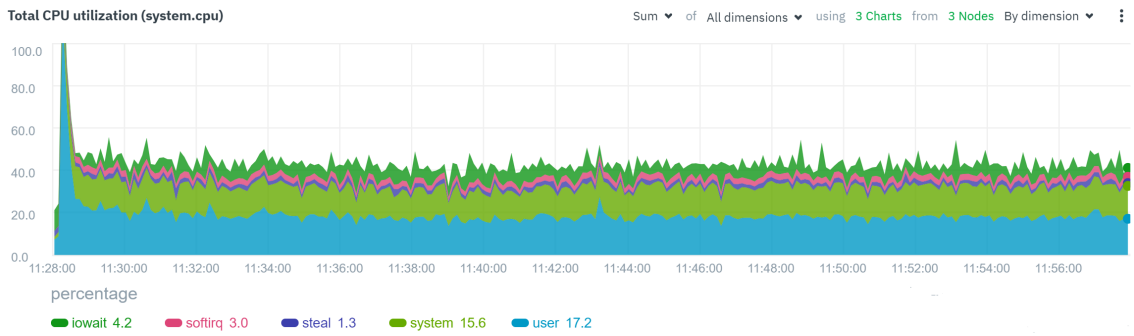


(d) Spark Disk IO

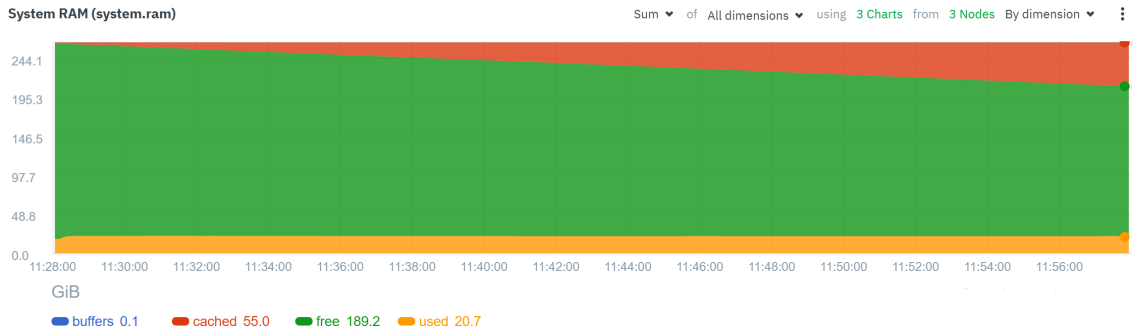


(e) Spark Network

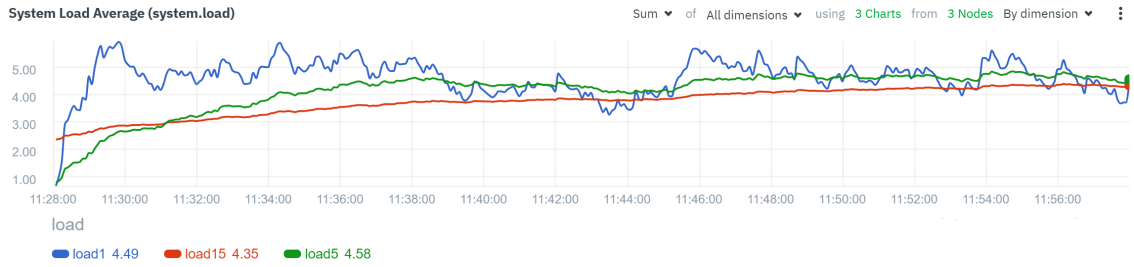
Figure 4.6 Spark Streaming Workload Resource Usage



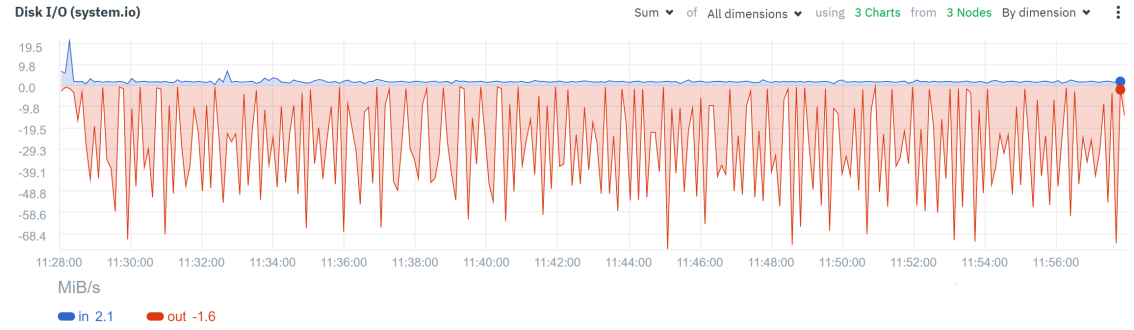
(a) Storm CPU Utilization



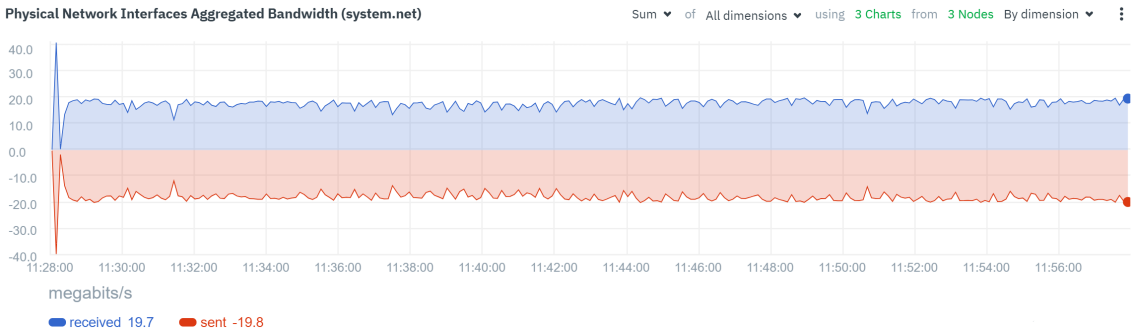
(b) Storm Memory Usage



(c) Storm System Load

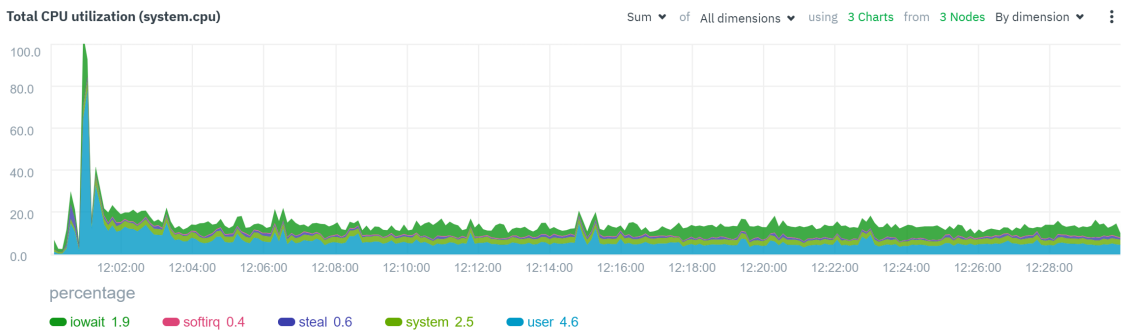


(d) Storm Disk IO

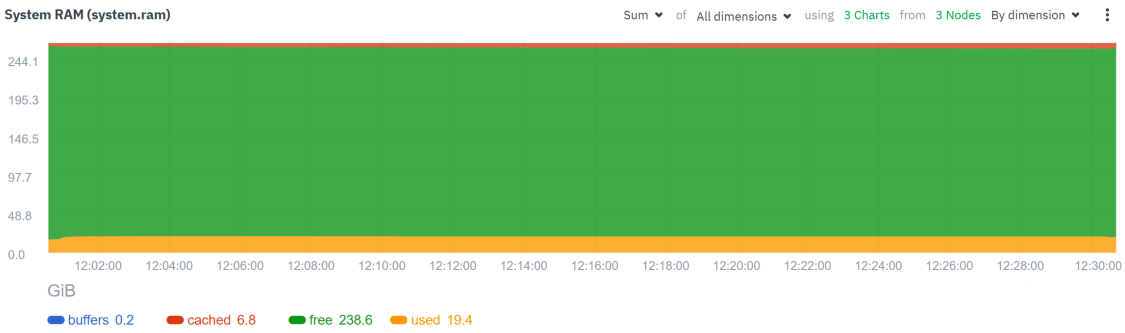


(e) Storm Network

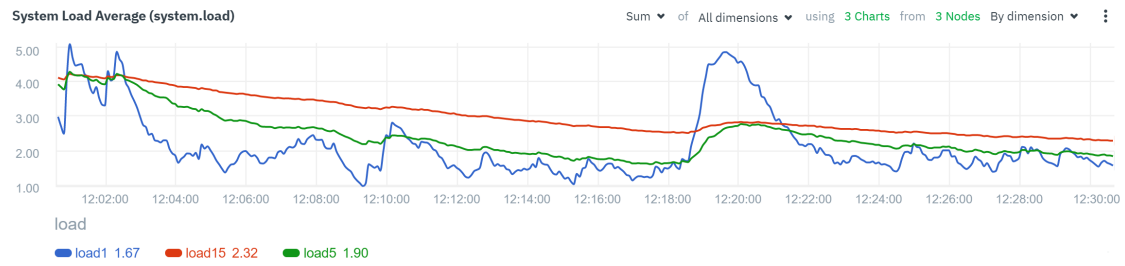
Figure 4.7 Storm Streaming Workload Resource Usage



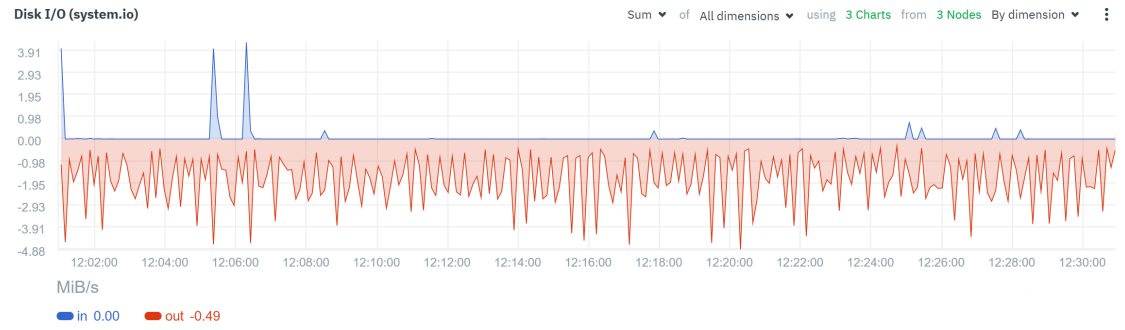
(a) Trident CPU Utilization



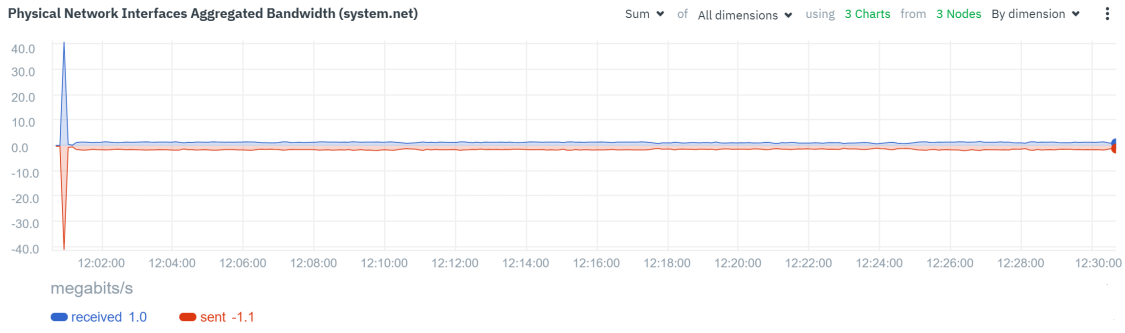
(b) Trident Memory Usage



(c) Trident System Load



(d) Trident Disk IO



(e) Trident Network

Figure 4.8 Trident Streaming Workload Resource Usage

CHAPTER 5 CONCLUSION

5.1 Summary

Our main contribution was to design a flexible, lightweight, customizable, and scalable analytics platform for health information systems. We proposed the Sandbox Architecture which is based on popular open-source technologies and on Kubernetes to provide a reliable layer for management and scalability. In addition, this deployment method makes an application more portable, simplifies dependency packaging, and facilitates repeatable and reliable build workflows.

This project also aimed to study big data tools and processes to choose the most efficient ones for this platform. To achieve this goal, we study mature stream and batch processing tools and the advantages and disadvantages of each.

In the last chapter, we used an HiBench benchmarking suite to validate Storm, Spark Streaming and Storm Trident from latency and resource usage perspective. Moreover, Another evaluation experiment was conducted among Hadoop and Spark in order to compare throughput results within three different profiles for Batch-ML purposes.

5.2 Limitations

There are also limitations for the research. There are other streaming tools available on the market. For instance, both Flink and Samza are gaining traction, there is a need to see some definitive values around their performance potential. Since the purpose of this study was not to conduct an exhaustive study of the available tools, but simply to demonstrate the application stream analytics on the domain of health. However, future works may be necessary to identify the optimal platform for the domain.

Another limitation is the benchmarking suite's implementation. While every effort was made to code the experiments according to best practices, each framework has nuances and quirks that can only be learned after a significant time. Looking through various framework-related forums revealed this fact. Given the variety of configuration options available within each framework, the scope of this study is relatively limited, with only some configurations tested. It was beyond the scope of this study to attempt to execute any further than what had already been done. Readers should be aware that this study and its findings only apply to the configurations specified in Chapter 4. Storm, as an example, allows developers to specify

the parallelism of individual spouts and bolts within a topology programmatically. This reveals how users can get better results with these frameworks; Many hours may be spent finetuning individual applications in production environments.

Lastly, the container environment in Kubernetes is not persistent by default. However, there are solutions to resolve this issue for the applications deployed in Kubernetes that need persistent storage to store data. These solutions need to be evaluated to see whether they perform well in various workloads and fluctuation of demands. There are also lots of concepts for better configuring and optimizing Kubernetes. This work only studies autoscaling and multi-tenancy. The evaluation of autoscaling methods in terms of various workloads was not also in the scope of this study.

5.3 Future Research

For the future research more studies for the other streaming application available on the industry is needed. Also, benchmarking of stream and batch computing frameworks on kubernetes can be a good practice to reveal how much resources is required for each workload and each streaming application. These framework need to evaluate in various cloud providers like Azure, AWS, OpenStack.

Another works can be also to compare performance between Storm Trident and Spark Structure Streaming.

The study on how to optimize setting up the application on kubernetes to reduce costs of resource usage. How to optimize application pod sizes to avoid wasting capacity. For instance, optimizing performance on spark to achieve best shuffle performance. Test various kubernetes autoscaling methods and compare in terms of various workloads latency and throughput, can be done.

REFERENCES

[1] M. Singh, V. Bhatia, and R. Bhatia, "Big data analytics: Solution to healthcare," in *2017 International Conference on Intelligent Communication and Computational Techniques (ICCT)*. IEEE, 2017, pp. 239–241.

[2] P. Galetsi, K. Katsaliaki, and S. Kumar, "Big data analytics in health sector: Theoretical framework, techniques and prospects," *International Journal of Information Management*, vol. 50, pp. 206–216, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0268401219302890

[3] G. Chen and M. Islam, "Big data analytics in healthcare," in *2019 2nd International Conference on Safety Produce Informatization (IICSPI)*. IEEE, 2019, pp. 227–230.

[4] S. Kumar and M. Singh, "Big data analytics for healthcare industry: impact, applications, and tools," *Big data mining and analytics*, vol. 2, no. 1, pp. 48–57, 2018.

[5] U. Akhtar *et al.*, "The impact of big data in healthcare analytics," in *2020 International Conference on Information Networking (ICOIN)*, 2020, pp. 61–63.

[6] A. Iyengar *et al.*, "A trusted healthcare data analytics cloud platform," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 2018, pp. 1238–1249.

[7] S. M. Krishnan, "Application of analytics to big data in healthcare," in *2016 32nd Southern Biomedical Engineering Conference (SBEC)*, 2016, pp. 156–157.

[8] S. P. Ahuja, S. Mani, and J. Zambrano, "A survey of the state of cloud computing in healthcare," *Network and Communication Technologies*, vol. 1, no. 2, p. 12, 2012.

[9] "Welcome." [Online]. Available: https://www.engineeringvillage.com/home.url

[10] L. Benhlima *et al.*, "Big data management for healthcare systems: architecture, requirements, and implementation," *Advances in bioinformatics*, vol. 2018, 2018.

[11] R. Chauhan and R. Jangade, "A robust model for big healthcare data analytics," in *2016 6th International Conference - Cloud System and Big Data Engineering (Confluence)*, 2016, pp. 221–225.

[12] W. Lin *et al.*, "A cloud-based framework for home-diagnosis service over big medical data," *Journal of Systems and Software*, vol. 102, pp. 192–206, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121214001368

[13] V.-D. Ta, C.-M. Liu, and G. W. Nkabinde, "Big data stream computing in healthcare real-time analytics," in *2016 IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, 2016, pp. 37–42.

[14] P. Johri *et al.*, "Vitality of big data analytics in healthcare department," in *2017 International Conference on Infocom Technologies and Unmanned Systems (Trends and Future Directions) (ICTUS)*, 2017, pp. 669–673.

[15] M. Sheeran and R. Steele, "A framework for big data technology in health and healthcare," in *2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON)*, 2017, pp. 401–407.

[16] R. Maheshwari *et al.*, "A machine learning based medical data analytics and visualization research platform," in *2018 International Conference on Current Trends towards Converging Technologies (ICCTCT)*, 2018, pp. 1–5.

[17] N. C. Taher *et al.*, "An iot-cloud based solution for real-time and batch processing of big data: Application in healthcare," in *2019 3rd International Conference on Bioengineering for Smart Technologies (BioSMART)*, 2019, pp. 1–8.

[18] T. Ali *et al.*, "The intelligent medical platform: A novel dialogue-based platform for health-care services," *Computer*, vol. 53, no. 2, pp. 35–45, 2020.

[19] A. Mohindra, D. M. Dias, and H. Lei, "Health cloud: An enabler for healthcare transformation," in *2016 IEEE International Conference on Services Computing (SCC)*, 2016, pp. 451–458.

[20] S. Landset *et al.*, "A survey of open source tools for machine learning with big data in the hadoop ecosystem," *Journal of Big Data*, vol. 2, no. 1, 2015.

[21] M. J. Kaur and V. P. Mishra, "Analysis of big data cloud computing environment on healthcare organizations by implementing hadoop clusters," in *2018 Fifth HCT Information Technology Trends (ITT)*, 2018, pp. 87–90.

[22] D. Chrimes *et al.*, "Interactive healthcare big data analytics platform under simulated performance," in *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(DASC/PiCom/DataCom/CyberSciTech), 2016, pp. 811–818.

[23] A. R. Rao and D. Clarke, "An open-source framework for the interactive exploration of big data: Applications in understanding health care," in *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 1641–1648.

[24] R. Mande, G. JayaLakshmi, and K. C. Yelavarti, "Leveraging distributed data over big data analytics platform for healthcare services," in *2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI)*, 2018, pp. 1115–1119.

[25] O.-C. Marcu *et al.*, "Spark versus flink: Understanding performance in big data analytics frameworks," in *2016 IEEE International Conference on Cluster Computing (CLUSTER)*, 2016, pp. 433–442.

[26] M. Khan, Salman, and N. Iqbal, "Computational performance analysis of cluster-based technologies for big data analytics," in *2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2017, pp. 280–286.

[27] S. Landset *et al.*, "A survey of open source tools for machine learning with big data in the hadoop ecosystem," *Journal of Big Data*, vol. 2, no. 1, pp. 1–36, 2015.

[28] "Production-Grade Container Orchestration." [Online]. Available: https://kubernetes.io/

[29] M. Kiran *et al.*, "Lambda architecture for cost-effective batch and speed big data processing," in *2015 IEEE International Conference on Big Data (Big Data)*. IEEE, 2015, pp. 2785–2792.

[30] "Kubernetes Autoscaling in Production: Best Practices for Cluster Autoscaler, HPA and VPA." [Online]. Available: https://www.replex.io/blog/kubernetes-in-production-best-practices-for-cluster-autoscaler-hpa-and-vpa

[31] "Helm." [Online]. Available: https://helm.sh/

[32] "Zeppelin." [Online]. Available: https://zeppelin.apache.org/

[33] "Apache Zeppelin 0.7.0 Documentation: Interpreters in Apache Zeppelin." [Online]. Available: https://zeppelin.apache.org/docs/0.7.0/manual/interpreters.html

[34] “Apache Thrift - Home.” [Online]. Available: https://thrift.apache.org/

[35] “Batch Processing Explained.” [Online]. Available: https://www.investopedia.com/terms/b/batch-processing.asp

[36] T. Shaikh, “Batch Processing — Apache Spark,” Jan. 2019. [Online]. Available: https://blog.k2datascience.com/batch-processing-apache-spark-a67016008167

[37] “What are advantages and disadvantages of batch processing systems,” Dec. 2012. [Online]. Available: https://www.itrelease.com/2012/12/what-are-advantages-and-disadvantages-of-batch-processing-systems/

[38] “Apache Hadoop.” [Online]. Available: https://hadoop.apache.org/

[39] “Hadoop vs. Spark: What’s the Difference?” [Online]. Available: https://www.ibm.com/cloud/blog/hadoop-vs-spark

[40] chandan prakash, “Choose Your Stream Processing Framework,” 05 2018. [Online]. Available: https://medium.com/@chandanbaranwal/spark-streaming-vs-flink-vs-storm-vs-kafka-streams-vs-samza-choose-your-stream-processing-91ea3f04675b

[41] G. Hesse and M. Lorenz, “Conceptual survey on data stream processing systems,” in *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*, 2015, pp. 797–802.

[42] “Apache ZooKeeper.” [Online]. Available: https://zookeeper.apache.org/

[43] “Apache Kafka.” [Online]. Available: https://kafka.apache.org/

[44] “Building a Streaming Analytics Stack with Apache Kafka and Druid.” [Online]. Available: https://www.confluent.io/blog/building-a-streaming-analytics-stack-with-apache-kafka-and-druid/

[45] “Apache Storm.” [Online]. Available: https://storm.apache.org/

[46] A. Toshniwal *et al.*, “Storm@ twitter,” in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, 2014, pp. 147–156.

[47] “Trident Tutorial.” [Online]. Available: https://storm.apache.org/releases/current/Trident-tutorial

[48] M. Zaharia *et al.*, “Apache spark: a unified engine for big data processing,” *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[49] "Spark Streaming vs. Structured Streaming - DZone Big Data." [Online]. Available: https://dzone.com/articles/spark-streaming-vs-structured-streaming

[50] "Differences Between RDDs, Dataframes and Datasets in Spark," Nov. 2020. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/11/what-is-the-difference-between-rdds-dataframes-and-datasets/

[51] "The most popular database for modern apps." [Online]. Available: https://www.mongodb.com

[52] L. DiFelice, "The CAP Theorem With Apache Cassandra and MongoDB," 08 2021. [Online]. Available: https://www.instaclustr.com/cassandra-vs-mongodb

[53] "Replication — MongoDB Manual." [Online]. Available: https://docs.mongodb.com/manual/replication/

[54] D. Abadi, "Consistency tradeoffs in modern distributed database system design: Cap is only part of the story," *Computer*, vol. 45, no. 2, pp. 37–42, 2012.

[55] V. Abramova and J. Bernardino, "Nosql databases: Mongodb vs cassandra," in *Proceedings of the International C* Conference on Computer Science and Software Engineering*, ser. C3S2E '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 14–22. [Online]. Available: https://doi.org/10.1145/2494444.2494447

[56] K. Anusha *et al.*, "Comparative study of mongodb vs cassandra in big data analytics," in *2021 5th International Conference on Computing Methodologies and Communication (ICCMC)*, 2021, pp. 1831–1835.

[57] "Realtime processing using storm-kafka- part2," Nov 2015. [Online]. Available: http://vishnuviswanath.com/realtime-storm-kafka2.html

[58] "Best Practice: Always 3 Nodes Minimum in a Cluster." [Online]. Available: https://www.scalecomputing.com/blog/best-practice-always-3-nodes-minimum-in-a-cluster

[59] "Compute Canada - Calcul Canada." [Online]. Available: https://www.computecanada.ca

[60] "Overview of RKE." [Online]. Available: https://rancher.com/docs/rke/latest/en/

[61] "Volumes and Storage." [Online]. Available: https://rancher.com/docs/k3s/latest/en/storage/

[62] "Kubernetes Autoscaling: 3 Methods and How to Make Them Great." [Online]. Available: https://spot.io/resources/kubernetes-autoscaling-3-methods-and-how-to-make-them-great/

[63] "Cluster multi-tenancy | Kubernetes Engine Documentation." [Online]. Available: https://cloud.google.com/kubernetes-engine/docs/concepts/multitenancy-overview

[64] "Three Tenancy Models For Kubernetes," Apr. 2021. [Online]. Available: https://kubernetes.io/blog/2021/04/15/three-tenancy-models-for-kubernetes/

[65] C. Wang, "Hibench," https://github.com/Intel-bigdata/HiBench, 2021.

[66] "nellaivijay/HiBench on GitHub." [Online]. Available: https://libraries.io/github/nellaivijay/HiBench

[67] "Hadoop vs Spark | Top 8 Amazing Comparisons To Learn," Aug. 2018. [Online]. Available: https://www.educba.com/hadoop-vs-spark/

[68] S. Huang *et al.*, "Hibench: A representative and comprehensive hadoop benchmark suite," in *Proc. ICDE Workshops*, 2010, pp. 41–51.

[69] J. Zhan, R. Han, and C. Weng, "Big data benchmarks, performance optimization, and emerging hardware," in *4th and 5th Workshops, BPOE 2014*. Springer, 2014.

[70] "Apache storm vs spark streaming - feature wise comparison," Nov 2018. [Online]. Available: https://data-flair.training/blogs/apache-storm-vs-spark-streaming/

[71] "Functional Comparison and Performance Evaluation of Streaming Framewo...," Oct. 2016. [Online]. Available: https://www.slideshare.net/HuafengWang/functional-comparison-and-performance-evaluation-of-streaming-frameworks

[72] "Apache Storm vs. Spark: Side-by-Side Comparison," Jul. 2021. [Online]. Available: https://phoenixnap.com/kb/apache-storm-vs-spark