WHITWORTH, JEFFREY N., M.S. Applying Hybrid Cloud Systems to Solve Challenges Posed by the Big Data Problem (2013)
Directed by Dr. Shanmugathasan (Shan) Suthaharan. 66 pp.

The problem of Big Data poses challenges to traditional compute systems used for Machine Learning (ML) techniques that extract, analyze and visualize important information. New and creative solutions for processing data must be explored in order to overcome hurdles imposed by Big Data as the amount of data generation grows. These solutions include introducing hybrid cloud systems to aid in the storage and processing of data. However, this introduces additional problems relating to data security as data travels outside localized systems to rely on public storage and processing resources.

Current research has relied primarily on data classification as a mechanism to address security concerns of data traversing external resources. This technique can be limited as it assumes data is accurately classified and that an appropriate amount of data is cleared for external use. Leveraging a flexible key store for data encryption can help overcome these possible limitations by treating all data the same and mitigating risk depending on the public provider. This is shown by introducing a Data Key Store (DKS) and public cloud storage offering into a Big Data analytics network topology.

Findings show that introducing the Data Key Store into a Big Data analytics network topology successfully allows the topology to be extended to handle the large amounts of data associated with Big Data while preserving appropriate data security. Introducing a public cloud storage solution also provides additional benefits to the Big Data network topology by introducing intentional time delay into data processing,

efficient use of system resources when data ebbs occur and extending traditional data storage resiliency techniques to Big Data storage.

APPLYING HYBRID CLOUD SYSTEMS TO SOLVE

CHALLENGES POSED BY THE

BIG DATA PROBLEM

by

Jeffrey N. Whitworth

A Thesis Submitted to
the Faculty of The Graduate School at
The University of North Carolina at Greensboro
in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Greensboro
2013

Approved by

____
Committee Chair

To my amazing wife Becca, for all her support, encouragement and love.

APPROVAL PAGE

This thesis written by JEFFREY N. WHITWORTH has been approved by the following committee of the Faculty of The Graduate School at The University of North Carolina at Greensboro.

Committee Chair ___________________________________

Committee Members ___________________________________

____

____
Date of Acceptance by Committee

____
Date of Final Oral Examination

ACKNOWLEDGEMENTS

I would like to thank my professor, Dr Shanmugathasan Suthaharan, for his guidance, support and direction over the past two years. He has helped me learn how to focus on research and provided many insightful discussions to help spur me forward. It has been an honor to work with him on not only my thesis, but research papers and projects.

I would like to thank my family. My wife, Becca, for endless support, understanding and encouragement. My kids, Eli and Nathan, for being willing to give up time with their dad so he could put in all the long hours. And all my family and friends, for making it possible to pursue this goal.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Big Data poses significant challenges to the field of Computer Science. The growth of data being generated and stored has significantly increased within the last decade [1]. This pattern has continued to accelerate and it is currently estimated that 2.5 quintillion bytes of data are generated every day [2]. This data is generated by an ever increasing number of sources ranging from personal electronic devices to network communication equipment to sensor networks. In addition to increasing data collection points, the data being collected is also becoming more complex and intertwined.

As a simple illustration, one can examine the challenges posed by Big Data that can be found by processing network firewall logs. As the available bandwidth of networks have increased, the amount of network firewall logs generated has also increased. This growth has outpaced the growth of traditional systems used for analysis, creating difficulties in gathering timely information from the network firewall logs.

The Information Technology industry has responded to challenges such as these by moving to a different model for computing resources that has been termed “cloud computing”. This shift leans heavily on being able to rely on “pooled” resources for small amounts of time and releasing some level of control of the underlying system performing the actual processing. Benefits of scale are inherited when these pooled systems can be shared across multiple organizations or users. Two immediate benefits are the ability to reach more efficient economies of scale and to quickly build complex systems without the upfront capital expenses typically required.

The increase in distributed processing as well as an increase in mobile devices has given way to increased data production. End users are able to interact with a cloud system from a mobile device by sending small amounts of data, or commands, which can then be executed on the cloud system. Results are returned to the end user with minimal data traffic. All the while the amount of data analyzed to return the results may be significant. The processing requirements of the end user device need only to meet the ability to send, receive and manipulate data. The complex data analysis can then be executed remotely. A good example of this can be found by examining parts of the Software as a Service product Twitter [7]. Take the example of topic trending within Twitter. A user is able to enter a small search term within a Twitter client. It is not the mobile client that produces the search results, but the data is sent to Twitter where data is then able to be searched for trends matching the user query. The client itself does not have to search through the 12+ Terabytes generated each data on Twitter [3], but just displays results found by using the Twitter API.

The amount of data being generated by end users within cloud systems is rapidly increasing [1]. As mobile devices become more accessible and mobile network bandwidth increasing [4], users are able to generate larger amounts of data and interact with cloud systems more readily, which in turn leads to increased data production. Such rapid increases in data production have necessitated new methods for data analysis to handle such an increase in data volume. Google helped pioneer the use of distributed systems, such as Hadoop [8], to process and analyze web graphs with its PageRank algorithm. Google has been able to find monetary value by analyzing the large amounts of data effectively.

An area where value has been found in data analytics can once again be illustrated by looking at Twitter and the example of a company releasing a new product. The ability to see interactions, comments and reactions in near real time throughout a user population in excess of 200 million, such as Twitter, not only provides indicators of success but also rapid analysis of the data that can occur within a time period that it is still relevant. Traditionally, this would require a significant investment to process over 12TB of data in order to deliver timely results. However, leveraging the trending data provided via Twitter's API allows this data to be analyzed within a large cloud ecosystem without the need for a large upfront investment.

Another instance where traditional systems have been outpaced by data growth is network traffic analysis. With such increases in bandwidth and mobile devices [4] the ability to examine network traffic logs in a timely manner has become more difficult. The ability to leverage supercomputer speeds to process data for short periods of time on demand offers efficiencies of scale as well as avoiding costly physical equipment.

Cloud services show promise to solve some of these general problems. Over the past several years the cloud platform has evolved from just being a term for server virtualization to an ecosystem that is evolved new software development methodologies as well as applications for end users that were previously reserved for businesses with the ability to make investments in expensive software products. Cloud services have also begun to change the paradigm used when sizing compute systems as they have the ability to scale much more efficiently. The Information Technology industry has begun shifting focus towards these cloud technologies and the definition of cloud services have been solidified, specifically around Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). The impact the above changes have on the Big Data Problem needs to be examined to determine if they offer advantages within compute systems used to process Big Data.

1.1 The Big Data Problem

Big Data is defined in a simplistic form by the characteristics of volume, variety and velocity [6]. Together, increases in these characteristics pose significant challenges to traditional compute systems, so much that traditional systems begin to break down and are unable to store and/or process the data. Data that causes this breakdown is defined as Big Data. System breakdown is often caused by the culmination of volume, variety and velocity such that the system is unable to store or process data within a meaningful timeframe. A simplistic illustration can be seen by comparing Figure 1 and Figure 2 below. This illustration uses a generic example of a queue that has incoming elements. Figure 1 shows normal being processed by a system. As time passes the queue length grows and recedes, eventually clearing all elements of the queue within an acceptable timeframe. Figure 2 illustrates Big Data system being processed by a system. As time passes the queue grows steadily. While the queue may process data as time elapses, it does not make significant progress towards clearing the queue. The system processing Big Data is unable to adequately clear the queue within a reasonable timeframe.

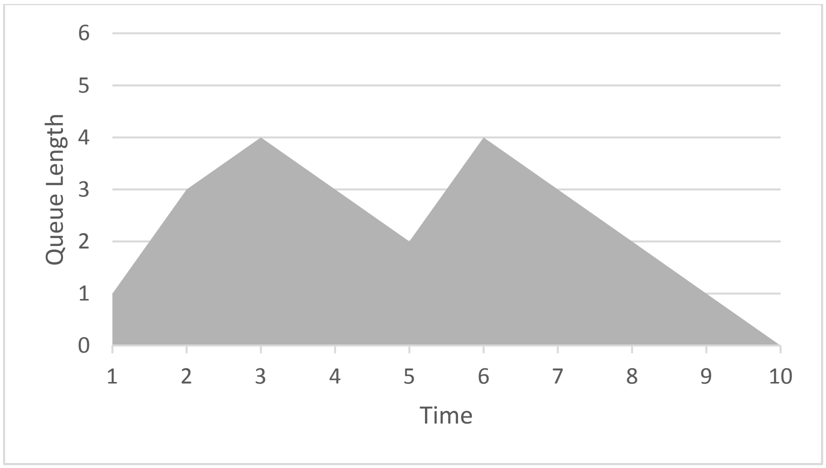


Figure 1. Normal Data Processing

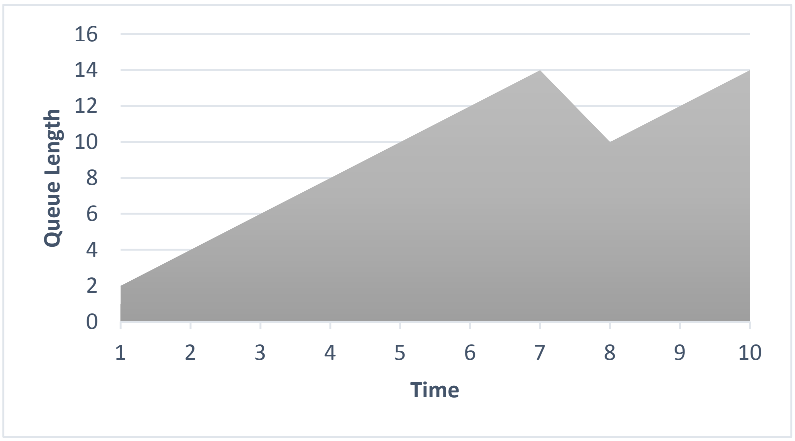


Figure 2. Big Data Processing

While the comparison between Figure 1 and Figure 2 may be overly simplistic, it helps illustrate the impact of volume, velocity and variety on a data system. This can be further shown by examining the specific example of Intrusion Detection Systems (IDS).

The evolving changes relating to increased bandwidth [1] and mobile device proliferation [4] have posed significant challenges to traditional Intrusion Detection Systems. These changes represent significant increases to the volume, variety and velocity of data and pose a threat to Intrusion Detection Systems and their ability to examine data in a timely manner. The very nature of IDS requires that they be able to quickly examine data and provide feedback to address possible active threats. This introduces a time constraint that a system must perform within in order to be effective. As data amounts increase – affecting velocity and volume – and device diversity increases – affecting variety – Intrusion Detection Systems become less effective because they cannot perform within their given time constraints.

1.2 Hypothesis

Traditional systems can be altered to overcome deficiencies that have been introduced with the Big Data problem while still maintaining the security benefits of a closed system. These alterations include leveraging distributed compute resources as well as cloud storage resources to overcome the problems outlined above. The security vulnerability introduced by data transmission and storage can be mitigated by a flexible key-based encryption technique that provides tradeoffs between time-delay, security strength and storage risks.

A simulation will be developed that will be used to re-create the Big Data problem within a previously proposed Intrusion Detection Network Topology [9]. The simulation will then be extended to include multiple cloud storage providers while still maintaining appropriate security controls required for various trust levels associated with public cloud providers. Experiments will then be run to show a formula exists that can be used to approximate data storage and retrieval times across a hybrid cloud system while still maintaining data confidentiality.

CHAPTER II

BIG DATA

First, it is important to understand what is meant by the term “Big Data”. Traditionally, Big Data is a term used to classify data when three characteristics of data - volume, variety and velocity – can no longer be effectively processed and stored [6]. Data meeting these characteristics is then termed Big Data. The three parameters – volume, variety and velocity – can be illustrated within 3 dimensional space, as shown in Figure 3 [9].

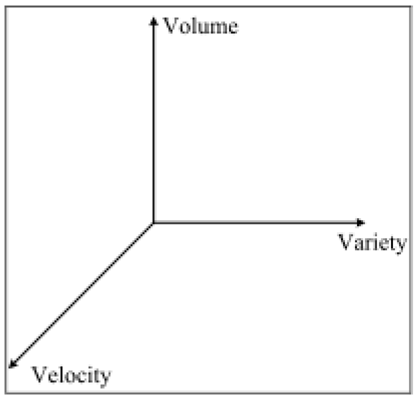


Figure 3. Big Data Parameters (V^3)

As the three parameters diverge data moves from been classified as normal data to Big Data. Continued divergence leads to further difficulties in both processing and storing data. Such volume exceeds traditional storage capacity. Such variety becomes increasingly complex to process. Such velocity occurs that the system cannot keep pace.

It is this combination that causes such difficulty for traditional systems – not necessarily any single parameter. Such high speed velocity of varying data occurs that the system cannot process and store the data without incurring an unbounded queue of data or a queue that can be cleared within a meaningful timeframe.

However, there are challenges with the traditional definition of Big Data. It can quickly be determined that the traditional definition does not define the point in which the actual capacity, processing capability or ability to handle velocity a system grow beyond a traditional system. This allows flexibility for the definition to change over time. Unfortunately, such a moving target may become difficult to grasp and remain relevant long term [11]. As shown by Moore's Law [12], data processing capability that is outside the grasp of traditional systems benefit but waiting for technological advancements.

The technology industry also asserts that a demand exists for new and efficient forms of processing this Big Data to gain insights and determinations [10]. This adds additional constraints on the definition of Big Data and offers reasoning that the simple V^3 definition of Big Data is actually too simplistic. Outside complexities, such as demand, influence the definition and are discussed in a later chapter.

2.1 Why Define Big Data

Overall, it is important to understand that data exists that cannot be processed by traditional compute systems within a timely manner [12]. It is also important to understand that a subcategory of this data, classified as Big Data, has specific challenges that may be overcome using non-traditional compute methods. As discussed previously, this occurs due to the challenges multiple characteristics of the data pose when combined together. Large increases in data generation [1] and the ability to gain valuable analytical insight are current problems facing Big Data. Defining Big Data will better help describe scenarios in which non-traditional compute systems may be leveraged to process data that a traditional system cannot.

It is also important to understand what Big Data is not. Data that has a lengthy processing time alone does not qualify as Big Data by traditional definition. Common use of the term Big Data has caused "super computer" worthy data to be classified incorrectly [11]. Specifically, data that does not exhibit the multiple characteristics is not Big Data because it is not just the size of the problem, but also its relation to data. A good example of this is comparing the problem of determining if a large number is prime to the problem of searching a social network. The problem of finding large prime numbers requires significant processing capabilities, but it does not exhibit the characteristics of Big Data. Searching a social network requires the ability to crawl a dataset that not only large, but growing in complexity, size and depth. It is important to make a distinction between problems that require supercomputer resources from those that require a Big Data approach.

There is also a specific point in which data crosses from being regular data and becomes Big Data. This point is interesting because it marks when a traditional system is no longer capable of being used and non-traditional systems must be employed. However, it also begs the question: if data can cross from being normal data to Big Data, can Big Data cross to being normal data? That is, must Big Data always be Big Data? Two options can be explored. First, the definition of traditional compute systems changes over time. Data that was once unable to be processed by a traditional system is now able to be processed. Second, if a characteristic of Big Data changes – say velocity of data can be slowed – the dynamic of V^3 changes. This can be related to Moore's Law [12] such that it may actually prove the same or better to wait for compute system performance to increase before beginning computation.

Defining Big Data allows determinations to be made about how to approach a problem and what type of system is best suited to solve such problem. Generalizing the traditional definition of Big Data (V^3) shows that the size of data, complexity of data and the speed at which data is collected have an important impact on how to build systems to process Big Data. As each of these characteristics constantly diverge from each other for an undetermined amount of time, systems must be capable of constant expansion and adaptation. In the case of an Intrusion Detection System (IDS), the system handles the variety or complexity of data as part of its native function. However, the velocity and volume of data pose a significant challenge to traditional IDS as data collection, devices and overall bandwidth increases. Understanding the nature of IDS Big Data allows systems to be designed to overcome these challenges.

2.2 Alternative Definitions

As stated previously, the traditional definition of Big Data (V^3) has shortcomings. Therefore, it is worth exploring alternate definitions and other influences that can be used to further the understanding of Big Data. As others have suggested [9], the traditional V^3 definition does not always take into account important characteristics of the data that is considered Big Data [9]. The argument has been made that the characteristics cardinality, continuity and complexity – referred to as C^3 provide a better method to describe Big Data as well as provide earlier detection. As shown in Figure 4, the relations of these three characteristics are similar to that of the V^3 relationships.

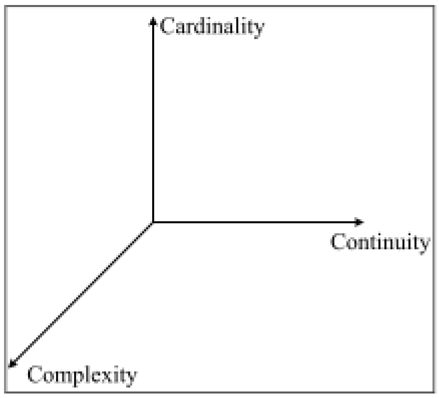


Figure 4. Big Data Parameters (C^3)

There are also other characteristics of Big Data that provide additional insights. While these are not strictly part of the definition of Big Data, they are common occurrences that may be taken advantage of when designing systems. One such characteristic is the ebb and flow of data, or the change of velocity at specific points in time. Common applications of Big Data often produce a large amounts of data within a small amount of time in such a manner that this burst of data quickly overwhelms a traditional compute systems ability. However, in this type of environment it may be possible to leverage a mix of traditional and non-traditional compute systems to appropriately meet the needs of varying velocity. It may be possible to determine a threshold that computation would change from a traditional system to a non-traditional system capable of handling Big Data, as seen in Figure 5. Control can then be returned to the traditional system assuming the non-traditional system has a higher cost.

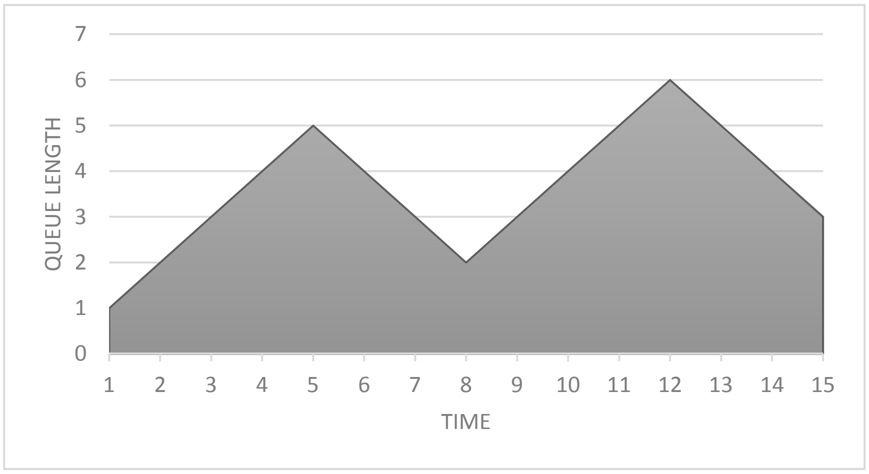


Figure 5. Hybrid Data Processing

2.3 Examining Volume, Variety and Velocity

Traditional compute systems fall short when being applied to Big Data for multiple reasons. Understanding those shortcomings is important when designing systems to be used in processing Big Data. These shortcomings can be shown by examining the limitations posed by Big Data on processors, memory, storage and the network bandwidth of a traditional computer system.

Data volume can be related to both memory and storage. Data volume that increases faster than a system can process data will eventually fill up memory and eventually any attached storage. Data variety can be related to processing capabilities. As data variety grows, a system must be able to adjust to accurately process the data. It stands to reason that growth in data variety will require additional processing cycles to compensate. As available processing cycles become scarcer data will continue to queue and the system will become further and further behind. Finally, data velocity can be related to both storage and bandwidth. As the speed in which data approaches increases more storage will be required. Bandwidth will also become saturated as velocity grows.

Another challenge with growing volume, velocity and variety is the inability to accurately plan to meet these needs. Sizing a traditional compute system requires understanding max system load. This is typically a known quantity that can be account for and calculated. However, the very nature of Big Data is constant growth as data is constantly being generated. This requires a system to be able to respond to growth in a flexible manner that does not disrupt data processing. A system must also be able to account for large data burst that can occur with Big Data. These diverse requirements require looking beyond traditional compute system to help solve problems posed by Big Data.

CHAPTER III

CLOUD COMPUTING: A TOOL FOR BIG DATA

Cloud compute systems have unique characteristics that become relevant when attempting to tackle the Big Data problem. By their very nature, cloud systems are designed to be scalable for specific amounts of time while later contracting. This is referred to as elasticity. A good analogy for elasticity is a rubber band. A properly sized system can operate with little stress, like a rubber band in a relaxed state. As the need grows, the rubber band stretches to accommodate the change in environment. This change may be short term or it may be a long term state of the rubber band. The rubber band is effective in maintaining an extended state. However, there is still a point that the band cannot stretch anymore without breaking. This is analogous to the limits found within cloud computing. However, the analogy breaks down in that cloud computing systems can constantly add capacity over time. It is not possible to add additional stretch capacity to a rubber band.

This elasticity may be the largest advantage cloud computing provides to attack the Big Data problem. A cloud compute system can expand processing, storage and bandwidth as the data volume, velocity and variety increase. More than just the ability to scale, cloud compute systems can expand and contract rapidly without intervention, providing a distinct advantage over traditional systems. Cloud compute systems grow to a much larger scale than a traditional system because they are built to be widely distributed across multiple data center and geographic locations. A traditional system is typically limited by physical location (building size, power, cooling, etc).

3.1 Cloud in Depth

Understanding the accepted definition of cloud computing helps provide clarity to characteristics of cloud compute systems. The generally accepted definition is outlined in a document from the National Institute of Standards and Technology. Mainly:

> Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. [13]

The definition later categorizes cloud computing into three varieties: Infrastructure as a Service, Platform as a Service and Software as a Service. These three varieties have been recognized by the industry as standards for cloud computing.

Software as a Service (SaaS) is one of the most common categories of cloud computing available today. It encompasses a large range of technologies including products that are provided by vendors for use within the cloud. Hosted email and collaboration services are good examples of Software as a Service offerings. SaaS is not the focus of this research.

Platform as a Service (PaaS) allows a provider to host client-created or owned software. Examples of PaaS include Microsoft's Azure web services, execution environment for client developed software [14], and Apache Hadoop [8], which will be discussed later in more detail. PaaS allows software to be directly executed on a cloud system instead of an Operating System. This is usually done through either a specific programming language or a variety of APIs. It is interesting to note that products can be developed upon a provider's PaaS by a user and then offered as SaaS to another user. This is actually a very common practice.

The final model is Infrastructure as a Service (IaaS). IaaS includes virtual machines, processing, storage and networking resources. A good example of IaaS is Amazons EC2 offering that allows consumers to run multiple Operating Systems on a hypervisor run by Amazon [15]. IaaS offers advantages over traditional infrastructures due to rapid deployment, elasticity and lower startup cost. Storage IaaS is part of the cloud platform that will be the focus of subsequent chapters.

3.2 Additional Challenges Posed by Cloud

Using a cloud compute system over a traditional compute system may help to tackle Big Data problems, but it also introduces complexities either not present in traditional systems or are given a higher priority when data is stored, traverses or is processed on an external system.

Network bandwidth constraints can pose significant challenges in a public cloud system. If data exists locally and must be sent to a public provider, this transaction becomes a possible bottleneck. Network bandwidth within the topology must be sufficient to handle these types of data transactions. Adding an external provider to the topology will introduce a level of uncertainty to the transaction speed and therefore this variable must be accounted for compared to data transfer within a local network topology.

Another significant challenge posed by using a public cloud provider is data security. While it is reasonable to assume that data in transport is easily secured by the use of accepted secure file transfer protocols or the use of SSL, data at rest still poses a challenge. Reputable public cloud providers state that data is encrypted at rest by their systems [14] [15]. However, most have provisions allowing data to be stored with 3rd party providers for which they are not responsible for data security. There also must be a level of trust with the public cloud provider since the provider is the one performing the encrypting and decrypting of the data at rest. This means the provider is also the one with the encryption keys. This poses a significant challenge that must be overcome.

Finally, there is an additional challenge posed when using a public cloud provider to store Big Data. Knowing data is available is important, especially if it is data that cannot be regenerated. This is fairly simple when dealing with normal amounts of data. One can simply perform a hash of the original data and perform subsequent hashes to verify data integrity. But by its definition, Big Data is significant in volume. Verifying that a public provider possesses the data that has been sent becomes a challenge. It is impractical to download and verify all the data exists using a hash. This would first require that a hash of the original data be hashed in its entirety. This is impractical with Big Data due to size. Additionally, this would require that subsequent integrity checks to download a copy of the data, which is also impractical due to the time and bandwidth required to download the entire data set. This poses a challenge when using a public cloud provider to store original copies of Big Data.

These challenges follow a general theme: lack of data control. Introducing a public cloud provider into a network topology will introduce influences to the system that are outside the control of the system. These challenges and risks are not typically present in a ML network topology and must be addressed when introducing them into a system using cloud resources. It is assumed that the benefit of utilizing the public cloud storage provider is worth the additional challenges and risks. Introducing the idea of a trust relationship between the system and the public cloud storage provider can aid in addressing these challenges. Ascribing a value to this trust relationship will allow appropriate mitigation of the risks introduced.

3.3 Public or Private

Cloud compute systems have multiple deployment models, two of which are termed public and private. Both of these models have advantages and disadvantages must be examined to choose the most appropriate model for a system.

A public cloud is defined as being "provisioned for open use by the general public" [13]. These offerings are built in a way that provisioning a service is simple and consumption standardized. This affords public providers some significant benefits such as the ability to provide on demand services and the ability to scale. These features are attractive to consumers as they significantly reduce the time it takes to build a system and ensures that clock cycles, storage or memory are not wasted by idling. Only resources required by the user are utilized by the user, nothing more. Public providers accomplish this by utilizing multi-tenancy and elasticity principles.

There are also drawbacks to a public cloud offering. These offerings must be more generic in nature in to meet the needs of the masses. Unfortunately, this can lead to reduced processing efficiencies. For example, public cloud service that provide Hadoop processing as a service must offer a generic interface into Hadoop. While the scalability and multi-tenancy features provide significant processing power and cost savings, the user is also forced to use a generic Hadoop implementation. It has been shown that using a finely tuned Hadoop instance for processing can lead to significant performance gains when compared to the default settings. In certain cases these savings are upwards of 70% [16], which is a compelling reason to still utilize local processing in order to tune configuration to best meet the needs of a specific application. Public offerings also require that all data be transferred to the public offering. This leads to data transfer latency, which can be significant for large datasets, as well as security concerns for the data.

The strengths of public cloud offerings are often the weaknesses of private cloud offerings. Private cloud systems lack the ability to scale to the same level of public clouds. They also require a significant upfront investment in capital expenses, operational expenses and have a much longer implementation time before becoming fully functional. Another disadvantage of a private cloud is long term wasted resources. A private cloud cannot take advantage of multi-tenancy to the same scale that a public cloud can. This makes it difficult to size a private cloud environment effectively. Traditionally a large investment is made in resources to ensure that peak loads can be processed. The side effect of this strategy is that resources go unused whenever the system is not at a peak load. However, keeping data and processing within a private cloud system allows processing to be tuned to specific workloads and security mitigations to be relaxed.

3.4 Hybrid

An alternative deployment model to public or private cloud computing is the hybrid model. This model is able to take advantage of the best of private and public offerings while minimizing the negative characteristics of both. Processing and storage can run locally within the private cloud as well as scale out to a public cloud when necessary. This model also requires a smaller initial investment and fewer wasted resources but can still take advantage of scalability features.

A Hybrid model poses some interesting possibilities. In the case of Hadoop, take a small cluster consisting of a single data node. Consider an additional node that behaves as a data node but actually sends data to a public Hadoop offering for processing. This could then be recursed, allowing for cloud chaining, as show in Figure 6.

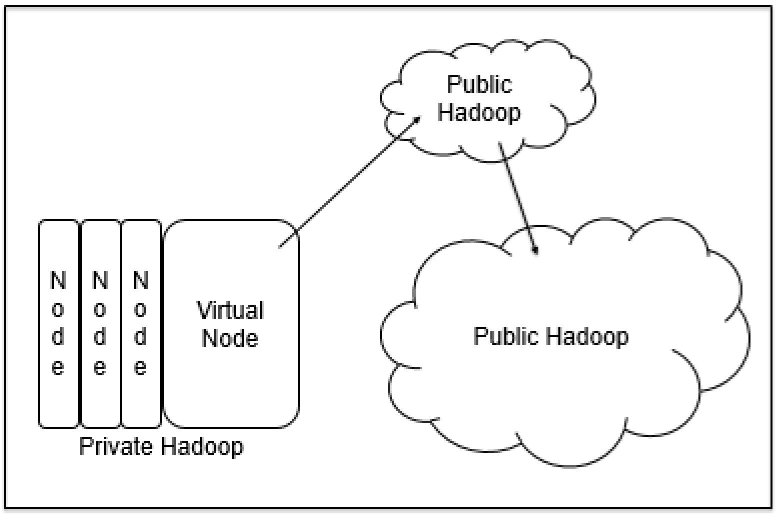


Figure 6. Hadoop Cloud Chaining

This type of design has some significant advantages. The user in this case does not need to be aware of any public offerings. The private cloud provider can then build a Hadoop system to meet the average need of users while extending to a public provider only when necessary. It would stand to reason that a public provider would be able to perform the same extension. This allows spikes in usage to be handled without having to build a system capable of handling all possible spikes.

An important question worth considering within a Hybrid model is when is it “worth” extending into a public offering? For example, a small Hadoop job is being executed but is configured to use a public cloud, it may actually take more time to open the connection, send and receive data from the public cloud than just to wait and process on the private cloud. It also opens the door to a cost based hybrid cloud computing model. This may aid in the determination of how much data to send to the public cloud. Previous work into cost-based analysis can be applied to this scenario [17].

Herodotou and Babu propose two import concepts relating to Hadoop cost-based optimization that have an application in determining hybrid cloud weights [17]. First, they propose an effective strategy for performing what-if analysis: Recursive Random Search (RRS). This method of analysis uses an algorithm to randomly sample and execute portions of a dataset. This technique can be applied in a scenario to help determine a weight that can be applied to a public and a private cloud to aid in performance calculations. Second, a framework to quantify performance in a MapReduce environment is proposed:

$$perf = F(p, d, r, c)$$

Performance is defined as a function of: program (p), input data (d), cluster resources (r) and configuration settings (c). This framework can be modified to provide weights to aid in determining how public and private cloud resources can be best utilized.

Before outlining a weighting scheme it is important to understand the key components that will have an impact on weights as well as the proper time to examine these weights and make a determination. The example below looks at processing a Hadoop dataset within a Hybrid offering.

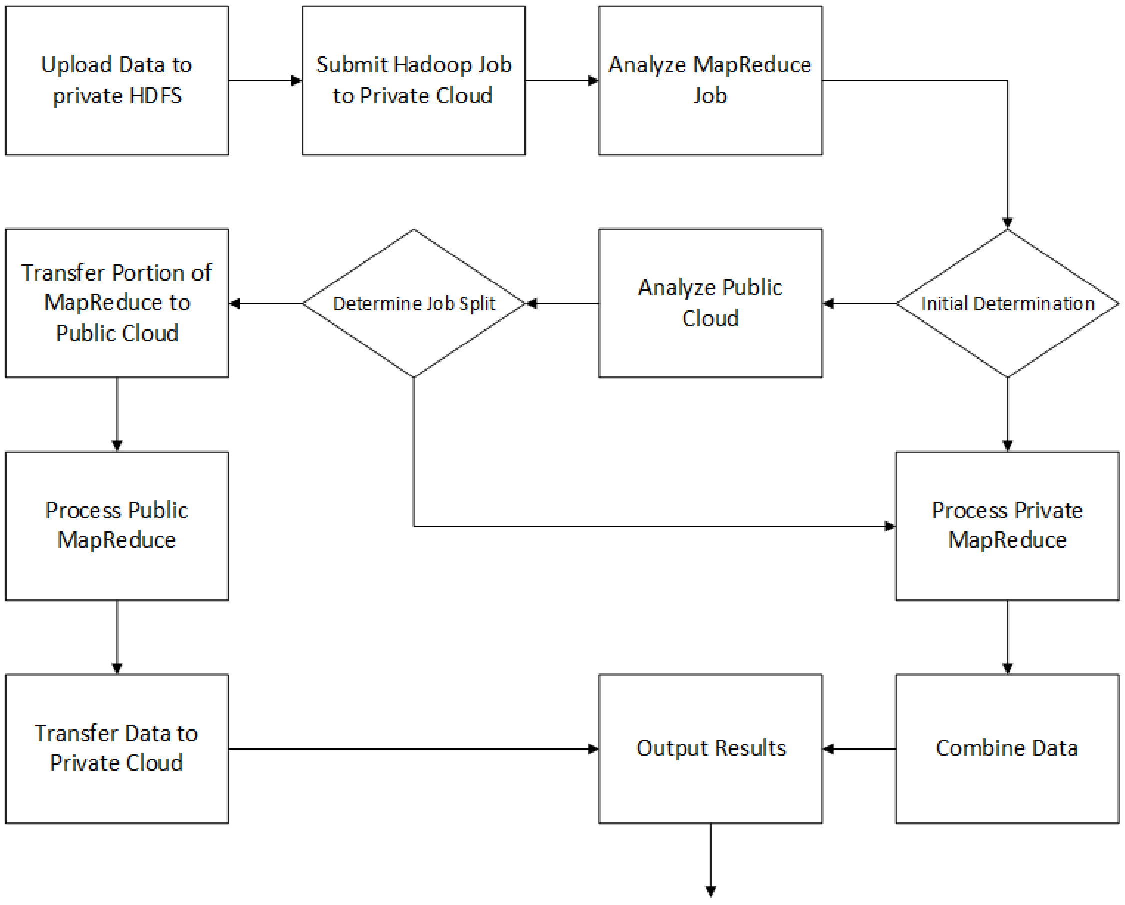

Upload Data to private HDFS
Submit Hadoop Job to Private Cloud
Analyze MapReduce Job
Initial Determination
Analyze Public Cloud
Determine Job Split
Transfer Portion of MapReduce to Public Cloud
Process Public MapReduce
Process Private MapReduce
Transfer Data to Private Cloud
Output Results
Combine Data

Figure 7. Hybrid Hadoop Process

The first step is to upload the entire dataset to the private cloud's HDFS. This dataset is denoted by d_T – the total dataset. Next, the Hadoop job is submitted to the private cloud. Before execution, an analysis is performed on d_T. Since the private cloud is a known resource a known value r_1 can be used to represent the private cluster resources. In this example, the cluster configuration (c) is considered static and therefore ignored. This analysis can be represented as: $\eta = F(p, d_T, r_1)$.

At this point an initial determination can be made. If η is so small that the private cloud resources will not be used in entirety, there is no need to utilize the public cloud and execution may be processed locally. However, if η is greater than the private cloud resources, further analysis will occur to take into account public resources.

Next, analysis is performed on the public cloud resources. Before performing full analysis, it is important to determine "spin-up" time, or the time required to utilize a public cloud. This can be represented by:

$$\gamma = F(d_{pub}, r_2, w_{pub})$$

Since the public cloud is known, w_{pub} is an assigned weight signifying the performance weight of the public cloud. w_{pri} represents the known private weight such that $w_{pub} + w_{pri} = 1$. This weight can also be used to determine the amount of data to be transferred to the public cloud, d_{pub}. Finally, r_2 represents public cloud resources.

If $\eta + \gamma < r_1$, it is determined that the Hadoop job should be executed only on the private cloud. Otherwise the job is executed in a hybrid mode. This is summarized in Table 1.

Formula	**Processing Location**
$\eta + \gamma < r_1$	Process on private cloud
$\eta < r_1$	Process on private cloud
$\eta + \gamma > r_1$	Process on hybrid cloud

Table 1. Hybrid Processing Determination

Introducing a public cloud adds several additional factors when determining total execution. In addition to the time to determine both η and γ, the actual cost of "spin-up" and "spin-down" must be added. "Spin-down" refers to the amount of time required to retrieve results through the gateway from the public cloud, clearing of any public cloud data and combining the results for output. It is also inferred that actual processing of d_{pub} is added. It can be shown that:

$$d_T * w_T = (d_{pub} + d_{pri}) * (w_{pub} + w_{pri})$$

or

$$d_T * w_T = d_{pri} * w_{pri}$$

The second equation holds when no public cloud is utilized. Table 2 compares the steps required for a private execution vs a hybrid execution.

Private Cloud	**Hybrid Cloud**
Local analysis	Local analysis
Spin-up analysis*	Spin-up analysis
Private execution	Spin-up
	Private execution
	Public execution
	Spin-down

Table 2. Hybrid Execution Steps

This type of analysis can prove valuable to a system that needs to determine at what point public cloud resources should be used. Answering this question is important since cost are typically incurred with bandwidth as well as public cloud consumption. Applying the concept of Recursive Random Search [17] is a viable means to produce this analysis.

CHAPTER IV

CLOUD APPLIED TO BIG DATA

Cloud computing has been used to solve several traditional computing problems but is also being applied to Big Data problems. Services like Amazon's AWS have been used to tackle Big Data problems in subject areas like Life Sciences [18]. Advantages of cloud computing, specifically scalability, are compelling reasons to look to public cloud services for solving problems such as human genome sequencing. The impact of services like Amazon's AWS have been shown in the cost and time to analyze the human genome [19] and the reduction in costs have been largely attributed to public cloud services.

The impact of cloud services on Big Data can best be shown by discussing how volume, velocity and variety are impacted. Scalability of a solution can provide the largest change to each of these Big Data parameters. For example, using a public cloud's Hadoop implementation to process Big Data allows the Hadoop instance to scale to meet additional data velocities and varieties, as well as the large amounts of volume being stored within Hadoop. Amazon's EC2 can scale to provide just over 240 TeraFLOPS of computational power [5]. This type of scalability provides the necessary processing power to overcome the challenges produced by Big Data while being able to use the services only when needed. This elasticity provides a distinct advantage over traditional compute system models. Specifically, cloud computing can be used within a Big Data Network Topology [9] to overcome challenges faced by traditional systems.

4.1 Proposed Model

The proposed design [9], shown in Figure 8, allows for processing to occur locally in order to take advantage of highly tuned processing within Hadoop. Storage within the system takes a hybrid approach to store data locally while also scaling into public cloud storage offerings to take advantage of the scalability available in public IaaS offerings. This ensures that data being collected can be stored even if the volume exceeds local capacity. However, it allows processing to remain within the local system.

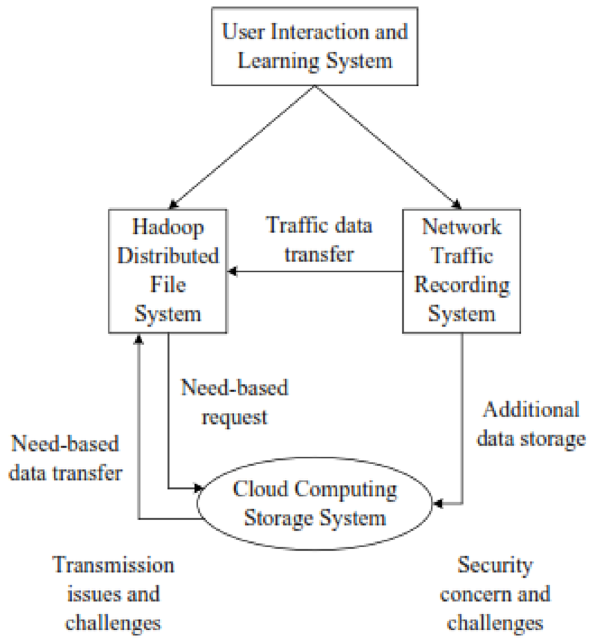

User Interaction and Learning System
Hadoop Distributed File System
Traffic data transfer
Network Traffic Recording System
Need-based request
Additional data storage
Need-based data transfer
Cloud Computing Storage System
Transmission issues and challenges
Security concern and challenges

Figure 8. IDS Network Topology for Big Data

This model also allows for the injection of intentional delay. Delay within a system is typically viewed as a negative, but in the case of this model it is used to address the volume and velocity of storage. It is assumed that at some point data is collected by the Network Traffic Recording System at a rate that would overwhelm an entirely private system in both allocated storage and the ability to efficiently process the collected data. As stated previously, it is important to process the data locally, so the Cloud Computing Storage System (CCSS) provides key functionality, one of which is the ability to delay data without exceeding allocated storage. Data is sent to a public cloud storage provider and can be requested when processing resources become available.

Due to the nature of Big Data it is possible that the NTRS may see an extremely high volume of data at times. The data delay imposed by using a public cloud storage provider may actually become beneficial by introducing delay into the system and acting as a buffer for the HDFS. Building a ML system that can withstand the high volume of data produced with Big Data can easily prove cost prohibitive. Intentional injected delay helps to solve that problem.

While it may seem like introducing this delay will create an ever increasing data backlog, this data can be processed when the data flow ebbs or the system is expanded to handle the increased load. By directing data away from the HDFS at an appropriate threshold, the HDFS can focus on computing data of a higher priority without the negative impacts of additional data strain while still allowing data to be processed locally.

Injecting delay into the system has added benefits to smaller scale research systems. A smaller scale ML research environment may see significant oscillation in data flow. However, it is very conceivable that storing the amount of data produced would incur significant cost to store locally. Leveraging an IaaS provider for data storage results in a buffer for processing as well as data storage in an on-demand and cost effective manner. This allows smaller scale research systems to use commodity hardware and avoid investing in expensive equipment that would sit idle when not processing data. Injecting intentional delay has benefit in scenarios where data velocity varies.

The use of the CCSS introduces challenges associated with public cloud solutions. Research into addressing security concerns has focused primarily on data classification [20]. Typically, data is either marked as safe for computation and/or storage with a public cloud provider or unsafe. This determines if that data is used within the public or private cloud. However, there are significant challenges with this strategy. First, it requires that data be marked as safe or unsafe. This process is either manual or relies on previously categorized data. Second, if a disproportionate amount of data is categorized as unsafe local resources will be overwhelmed. It would be possible that all data is categorized as unsafe and therefore the system would cease to be a Hybrid cloud.

Therefore, the CCSS assumes that all data has a constant level of sensitivity and it is the public cloud provider characteristics that determine how the data must be protected within the storage provider. This is accomplished by assigning a trust level to providers and taking necessary steps to mitigate risks.

4.2 Determining Trust

While public providers may attempt to mitigate these risks, some level of risk remains due to the loss of total control by the user. Therefore, some level of trust between a consumer and provider must be established. In general, greater trust means the user must mitigate less risk. Characteristics of a provider can be ascribed a weight that is associated with the Big Data being analyzed. Together these characteristics and weights can be combined to determine a trust level with a specific provider. This serves as a model that can be used to determine the level of encryption needed to safely store data. When working with large amounts of data associated with Big Data analytics minimizing computational overhead is valuable. Significantly more complex data encryption on high volume, high velocity data would lead to inefficiencies.

In order to facilitate a trust determination a system of identifying characteristics of both the data and storage providers is used. A weight is ascribed to each of these characteristics that reflect a mitigation level. While this system is rudimentary, it is meant to serve as groundwork for future research into automatically determining these weights with various providers.

Example characteristics include previously mentioned multi-tenancy and elasticity as well as concepts such as securing deleted data, the use of 3rd party data storage providers, adherence to safe harbor, etc. Depending on the type of Big Data being stored with an IaaS provider these characteristics may become valuable when determining the feasibility of levering a particular IaaS provider for data storage. Due to the ever-increasing number of possible characteristics one must consider when developing a trust level with a provider, this model remains generic enough to accommodate any number of characteristics. Unfortunately several characteristics currently require human intervention to make a determination. This is a challenge posed by the cloud services industry due to lack of standards and transparency.

Each characteristic can then be ascribed a weight proportional to the importance of the characteristic in relation to the dataset being analyzed. These weights can be controlled within the User Interaction and Learning System so they can be adjusted depending on the data being analyzed. For example, a public data set common to Big Data may have very low weights while analysis of secure network communications may have much higher weights. Health related data or Personally Identifiable Information may have an even higher weight. This model is flexible enough that the weights of each characteristic may be changed depending on the type of dataset being analyzed.

Calculating risk in such a dynamic ecosystem poses significant challenges. This research does not attempt to identify all possible risks but instead put forth a more abstract model that can account for an unknown number of risks. As a base, the below is used to identify the risk associated with a single characteristic and the weight ascribed by producing a value $\alpha = c * weight$. Three parameters influence the efficiency of this model: α, c and $weight$. The relationship between these three parameters can be explained by a simple non-linear model. The equation explains the relationship between the risk level (α) and importance level ($weight$) for a fixed mitigation level. Similarly, it also explains the relationship between the risk level (α) and mitigation level (c) for a fixed importance level ($weight$).

The value c refers to an identified mitigation level between 0 (full mitigation) and 10 (no mitigation) performed by the provider. At this time the level is determined by manual analysis of the IaaS provider such as in place agreements, service terms and peer's understanding and may be different between multiple providers. The $weight$ refers to an importance of the characteristic as it relates to the Big Data, with 0 being no importance to 1 being the highest importance. Therefore an α of value 0 would have no risk while a value of 10 would be the highest risk. The value β is an ensemble of all risk levels α, as shown by: $\beta = \alpha_1 + \alpha_2 + ... + \alpha_n$

A higher β value implies a lower overall trust associated with a provider in the CCSS. Since the value of β is the sum of an unknown number of α values, there is no upper bound defined for the possible risk. It can also be concluded that a completely trusted environment, such as a private cloud storage system, may have a β value of 0.

The value β can then be used in association with identified encryption types (ETypes) to determine an appropriate level. Each encryption type has a max acceptable β value, as seen in Table 3.

Max β Value	**EType**
0	1
1 - 3	2
4 - 14	3

Table 3. EType Values

The example in Table 3 shows an EType of 1 that is only appropriate for use when the β value is 0, while the EType of 2 is appropriate for β values 0-3. The EType 3 is appropriate for all β values up to 14. A β value greater than 14 would mean that the provider cannot reasonably be used. It is important to note that higher ETypes are appropriate for use with all previous ETypes. This allows for both the appropriate level of encryption to be used as well as flexibility when selecting efficient algorithms. It is assumed that lower ETypes will allow Big Data analysis to be performed quicker due to decreases in computation complexity.

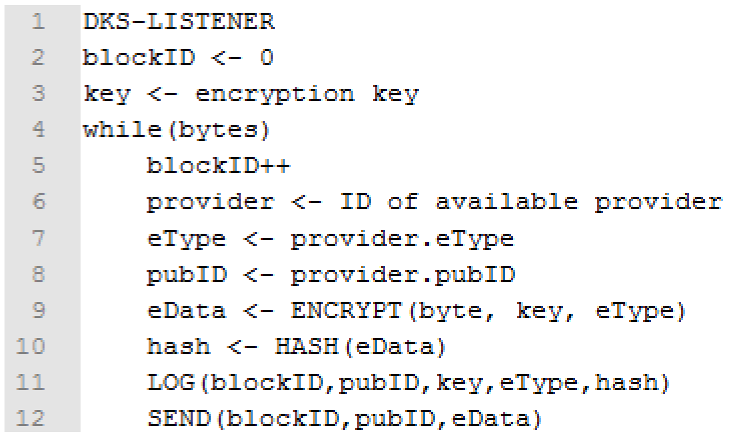
4.3 The Data Key Store

It is possible to expand on the Network Traffic Recording System (NTRS) in Figure 8 to provide mitigations of security concerns and challenges when storing data in a Cloud Computing Storage System (CCSS). This is done by including a Data Key Store (or DKS) as part of the NTRS that manages data sent to the CCSS. An additional benefit of the Data Key Store model is that it can be used to provide other functions including proofs of retrievability. Being able to periodically prove the ability to retrieve data becomes increasingly important as datasets grow and cannot efficiently be examined without significant burden [21], as in the case of Big Data. However, in an ecosystem using IaaS for data storage, it is important to be able to prove a provider actually possesses the data and not just a placeholder. While the Data Key Store model can be used solely to handle encrypting data at appropriate levels, it can be extended to provide a form of retrievability proof.

The Data Key Store (DKS) is used to manage the data stored with the IaaS provider as well as the associated keys required for encryption and decryption of data. The DKS is integrated with the NTRS in Figure 8 as a mechanism to move data to the

IaaS provider while overcoming security concerns and challenges. Another possibility of the DKS that may prove valuable when working with Big Data is the ability to use multiple public cloud storage services. Data can even be striped across multiple storage providers allowing added resiliency for the data stored with an IaaS provider.

Each block stored has a 5-tuple value that includes blockID, PubID, Key, Hash and EType. These values are used to track a block of data (blockID), the storage provider (PubID), the encryption key (Key), a hash of the data (Hash) and the encryption type (EType). This allows for flexibility to store data with multiple vendors, each with their own associated risk level and corresponding encryption type. An algorithm for this process is shown in Figure 9.

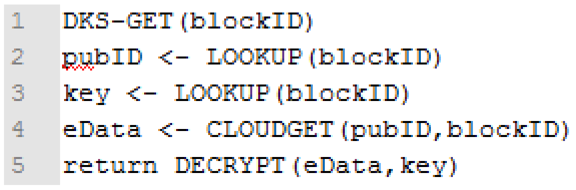


```
DKS-LISTENER
blockID <- 0
key <- encryption key
while(bytes)
    blockID++
    provider <- ID of available provider
    eType <- provider.eType
    pubID <- provider.pubID
    eData <- ENCRYPT(byte, key, eType)
    hash <- HASH(eData)
    LOG(blockID,pubID,key,eType,hash)
    SEND(blockID,pubID,eData)
```

Figure 9. DKS-LISTENER Algorithm

The algorithm listens for incoming bytes of data from the NTRS. As each byte is received it is encrypted using the predefined key and encryption type for the public provider being used. It is then hashed and metadata is stored within the DKS consisting of the blockID, the pubID, hash and encryption key used. The encrypted data is then sent to the corresponding storage provider.

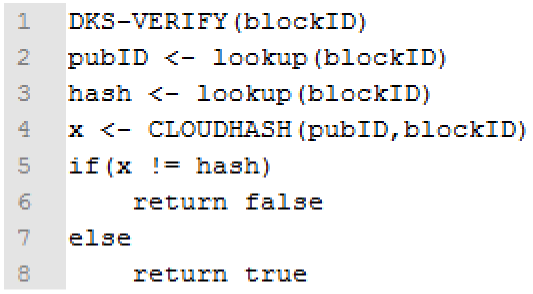
The HDFS can retrieve data from the CCSS through the DKS. This is done using a retrieval algorithm found in Figure 10.



```
DKS-GET(blockID)
pubID <- LOOKUP(blockID)
key <- LOOKUP(blockID)
eData <- CLOUDGET(pubID,blockID)
return DECRYPT(eData,key)
```

Figure 10. DKS-GET Algorithm

There is also a verification algorithm that can be used to verify the hash of the encrypted data in order to prove the provider contains the data. The DKS-VERIFY algorithm is similar to the DKS-GET algorithm and is shown in Figure 11.



```
DKS-VERIFY(blockID)
pubID <- lookup(blockID)
hash <- lookup(blockID)
x <- CLOUDHASH(pubID,blockID)
if(x != hash)
    return false
else
    return true
```

Figure 11. DKS-VERIFY Algorithm

The implementation of the CLOUDHASH function can be either local to the DKS or it can be implemented within the public cloud provider. Implementation within the provider allows the data to remain with the public provider and the processing required to hash that data to occur within the public cloud provider.

4.4 Proving Big Data Retrievability

As mentioned earlier, the DKS contains information that can be used in a basic proof of retrievability, while laying the foundation for more complex proofs [21]. The ability to prove an IaaS provider actually contains the data stored becomes more difficult as data size increases. Due to the loss of data control and varying trust levels it becomes more important to verify data is actually stored and not just a stub file or pointer to a 3rd party storage vendor that may no longer contain the data. However, with such large files, simply retrieving the data from the IaaS provider is not feasable.

Since the DKS breaks data into blocks, it is also trivial to store a hash of the block. It is much more efficient for the DKS to periodically retrieve a single block and verify the hash, as seen in the DKS-VERIFY algorithm. While this is much more trivial than the model shown by Juels and Kaliski [21], it does provide a simple proof of retrievability that can be extended with further research.

CHAPTER V

RESULTS

A simulation can be shown of the Data Key Store and multiple cloud providers representing the Cloud Computing Storage System. This requires implementation of the DKS-LISTENER, DKS-GET and DKS-VERIFY algorithms. The following implementation is written in Java and shows results obtained by running a benchmark analysis and multiple executions storing data in a hybrid cloud system. The hybrid cloud used in the simulation consists of an embedded Java database, H2 [22], acting as the private cloud storage solution. The first public cloud provider is a remote database server running MySQL Community Edition [23]. The second public cloud provider is Microsoft Azure Blob storage [14].

5.1 DKS-LISTENER

The DKS-LISTENER is used to read a byte[] input stream, in this case a FileInputStream. This simulates the Network Traffic Recording System shown in Figure 8. The DKS then determines the eType based on available storage providers and encrypts the data. A SHA-1 hash of the encrypted data is calculated for later verification. This information is then logged in a local database. A unique block id and the encrypted data are then sent to the provider.

Artificial limitations are later put into place to simulate exhausting all available storage capacity within each provider. This is achieved by implementing a Provider.IsFull() method that runs before writing any byte[] to a provider. If Provider.IsFull() is false, the data is written and the used capacity for the provider is incremented. If Provider.IsFull() is true, the next available provider is chosen.

Encryption method is chosen based on the determined EType value. These EType values are derived according to: $\beta = \alpha_1 + \alpha_2 + ... + \alpha_n$. As an example, a β value is calculated for each provider α values are determined based on the risks and ascribed weights shown in Table 4 below.

Risk Level	**Risk**	**Ascribed Importance (*weight*)**
α_1	Multi-Tenancy	1
α_2	Data Location	0.5
α_3	Physical Access Restrictions	0.7
α_4	Shared Storage	1

Table 4. Simulation α Values

The example α values can be used to determine a β value of the provider. Let β_1 represent the total risk level for the private cloud storage. In this example case, multi-tenancy and shared storage are the highest weighted characteristics for this type of data, followed by physical access restriction and data location. Table 5 illustrates the risk levels for the private cloud provider in the simulation.

Risk Level (α)	**Ascribed Importance** (***weight***)	**Mitigation Value** (c)	***weight * c***
α_1	1	0	0
α_2	0.5	0	0
α_3	0.7	0	0
α_4	1	0	0

Table 5. Private Cloud Risk Levels

Since the private provider is fully mitigated the total risk level for the private cloud provider (β_1) is 0. Table 6 illustrates the risk levels ascribed to the public cloud remote database storage– denoted by β_2.

Risk Level (α)	**Ascribed Importance** (***weight***)	**Mitigation Value (c)**	***weight * c***
α_1	1	3	3
α_2	0.5	2	1
α_3	0.7	2	1.4
α_4	1	8	8

Table 6. Public Cloud Risk Levels

Since the remote database is running in a known, but not necessarily controlled environment, some mitigation must be taken. For example, the database location is a known physical location with controlled access, therefore α_2 and α_3 both have lower values. However, the server is a shared server that allows others to have elevated access rights, so α_4 and α_1 have higher values. The β_2 value for the public cloud database in this example is 13.4.

Finally, Microsoft Azure Blob storage is the third storage provider. Associated risk levels for the Azure Blob storage service are shown in Table 7.

Risk Level (α)	**Ascribed Importance (*weight*)**	**Mitigation Value (c)**	***Weight * c***
α_1	1	10	10
α_2	0.5	6	3
α_3	0.7	1	0.7
α_4	1	8	8

Table 7. Microsoft Azure Blob Risk Levels

In this example, the data storage is a multi-tenancy location so it has the highest α_1 value. The storage is shared, but in this case the vendor is transparent about how it encrypts data at rest, so α_4 is given a value of 8. The exact physical data location is unknown, but certain assurance are given about the region and country data is located within, scoring an α_2 value of 3. Finally, following a growing trend in the industry, access is extremely limited within Microsoft Azure data locations, scoring a 0.7 for α_3. This leads to a β_3 value of 21.7.

These β values are then used to ascribe an appropriate EType base off a predefined table for this dataset. The EType table used in the experiment is shown in Table 8.

Encryption Type (EType)	**Max β Value**	**Encryption Algorithm**
1	0	None
2	15	3DES
3	35	AES-256

Table 8. Simulation ETypes

5.2 DKS-GET

The DKS-GET algorithm, shown in Figure 8, is used to retrieve specified blocks. The input is a blockID, or range of blockID's, that are first looked up in the local log database. The log database holds the storage provider ID as well as the encryption key. The provider ID and blockID are used to retrieve the encrypted block from the provider. This block is then decrypted and returned.

Since the decryption process happens within the Data Key Store and not on the provider, this computational time can be taken out of the performance calculation of the cloud provider and becomes a function of the Data Key Store itself. Figure 12 shows the decryption time for each of the storage providers.

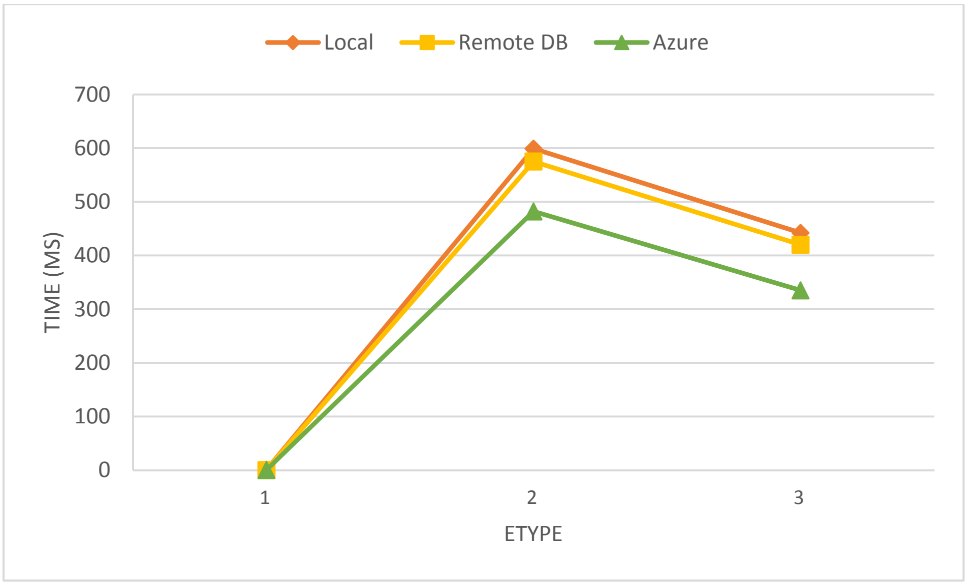


Figure 12. Decryption Benchmarks

Figure 12 shows the decryption time for each of the three providers to decrypt the same blocks. This experiment was run with EType 1, 2 and 3 on all providers. There is no significant variance between providers, which supports the proposed assumption that decryption time is independent of the cloud storage provider.

5.3 DKS-VERIFY

The DKS-VERIFY algorithm, shown in Figure 9, has similar properties to DKS-GET. The input is a given blockID that is then used to lookup the hash and provider ID in the local log database. The CLOUDHASH method in line 4 is used to retrieve the encrypted block and return the SHA-1 hash. It is possible that the CLOUDHASH method could be implemented with the provider itself, allowing the provider to return a hash calculated by or on the provider. This may be of use if a more complex data possession or retrievability proof is required [21].

The DKS-GET algorithm could also be extended to use DKS-VERIFY to perform a verification that the block stored with a provider is not corrupted before attempting a decryption.

5.4 Performance Benchmarking

A set of performance benchmarking is needed to understand the performance capabilities of the storage providers in relation to each other. This is also used to determine the r values for each provider in the performance formula $perf = F(p, d, r, c)$. These r values are also used calculate the γ values for the public storage providers and the β values for the local storage provider. These values are important in understanding cost of using a public storage provider and in determining the point in which using a public cloud provider is of value.

To determine benchmarks, an experiment was run with the DKS-LISTENER and the DKS-GET methods. A sample data file, the text of *Les Miserables* [24], was used in all experiments. This file is 3,245KB in size. Since the chosen block size is 1024 bytes, this translates into each run containing 3,245 blocks of data. Figure 12 shows the average of three runs with each provider. Each run used a single provider with an EType of 1, or no encryption. Appendix A contains results of each individual run.

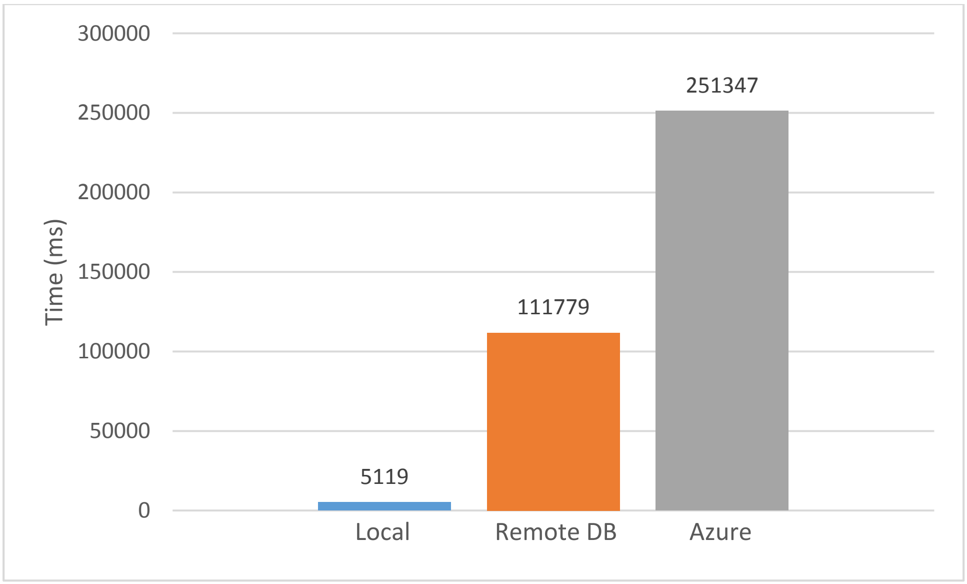


Figure 13. DKS-LISTENER Benchmarks

As seen in Figure 13, the local embedded database storage performed roughly 20x faster than the remote database storage provider performed and roughly 50x faster than the Azure Blob storage on average. The remote database storage provider performed roughly 2.25x faster than the Azure Blob storage on average. Changing the EType has no significant impact on the performance.

The average time to retrieve the same data using DKS-GET are shown in Figure 14. The local embedded database storage performed roughly 10x faster than the remote database storage provider performed and roughly 20x faster than the Azure Blob storage on average. The remote database storage provider performed roughly 2x faster than the Azure Blob storage on average. Changing the EType has no significant impact on the performance.

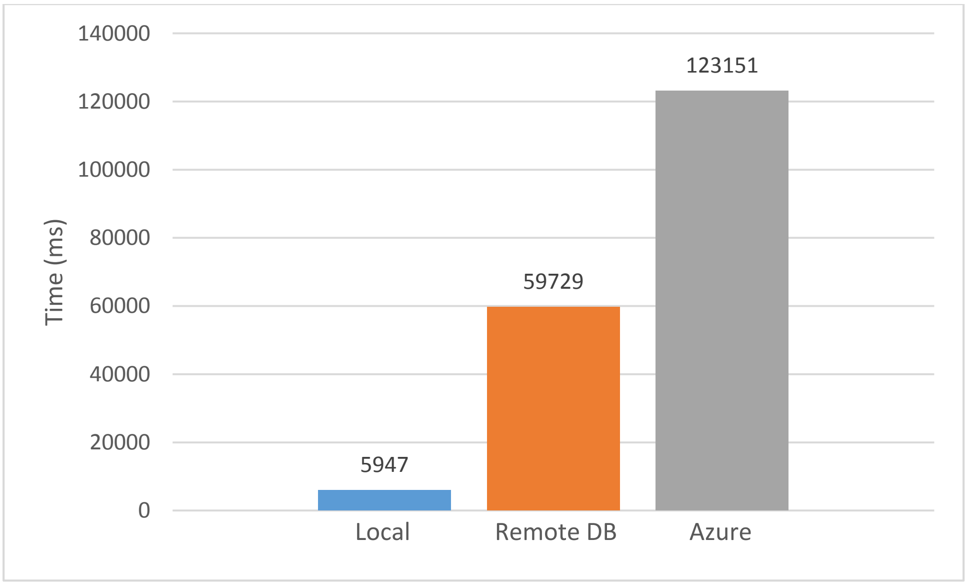


Figure 14. DKS-GET Benchmarks

The time to the retrieve benchmark data from the local provider was slightly higher (828ms) than the time to write the data. This is most likely attributed to the database implementation and retrieval operations. However, both the remote database and Azure Blob storage performed significantly faster when retrieving the data than when sending the data. It is possible that network bandwidth has an impact since both of these providers require the data to traverse networks. It is also possible that the data retrieval implementations in MySQL and Microsoft Azure Blob storage are significantly (almost 2x's) more efficient than the write operations.

Overall, the benchmarks performed as expected. Local storage performs much faster than the public cloud storage providers. This proves that additional considerations should be taken when large datasets are generated locally and are sent to a public cloud provider for processing. Based on the benchmarks this data traversal may take fifty times longer than using local storage. Therefore the additional processing power of a cloud provider must be significantly more powerful to outweigh the cost of remote storage. However, in the case of Big Data, local storage may not be enough so using a public cloud provider may be a necessity.

5.5 The Hybrid Cloud Experiment

The performance benchmarks provides the basis needed to determine the *perf* value for each provider. The equation *perf = F(p, d, r, c)* can be used to determine this performance. As proposed earlier, *perf* is a function of four values: *p* – the program, *d* – the data, *r* – the cluster performance and *c* the configuration. This can be adapted from the proposed performance of computation resources to storage resources by treating *p* as a constant since the program is the same among all providers: data storage. The *c* value can also be considered a constant since the configurations during the benchmarking are the same, namely block size and encryption type. *d* represents the total number of blocks of a particular dataset. In the case of the benchmarking, this would be 3,245. The *r* value can be defined as follows:

$$r_x = \frac{d_T}{SEND_x + GET_X}$$

d_T represents the total amount of data. $SEND_x$ represents the time (in milliseconds) required to send d_T to the storage provider, or the DKS-LISTEN algorithm. GET_x represents the time (in milliseconds) requires to retrieve d_T from the same storage provider.

Let r_1 represent the resources in the local storage provider. The value of r_1 is:

$$3{,}245/(5{,}119 + 5{,}947) = .2932$$

Let r_2 represent the resource of the remote database storage provider. The value of r_2 is:

$$3{,}245/(111{,}779 + 59{,}729) = .0189$$

Let r_3 represent the resource of the Microsoft Azure Blob storage provider. The value of r_3 is:

$$3{,}245/(251{,}347 + 123{,}151) = .0087$$

These values provide a good description of storage resource performance that is in line with the time required to send and retrieve data to respective providers. The higher r value the better storage resource performance. The performance function $perf = F(p, d, r, c)$ can now be defined as $perf_x = F(d, r_x)$ since p and c are constants. This can be used to estimate the performance of a provider x on d data blocks. If the relationship between d and r_x is d/r_x then $perf_1 = d/r_1 = 3{,}245/.2932 = 11{,}067$. This proves correct as the time (in milliseconds) required to send and retrieve 3,245 block to the local storage provider, as shown in the benchmark results. This can be further shown with:

$$perf_2 = d/r_2 = 3{,}245/.0189 = 171{,}693$$

$$perf_3 = d/r_3 = 3{,}245/.0087 = 372{,}988$$

where $perf_2$ is the resource performance of the remote database storage provider and $perf_3$ is the resource performance of the Microsoft Azure Blob storage. These performance values are also in line with the benchmark results. It stands to reason that $perf_x$ can be used to estimate performance time of each provider with varying d values. For example, given a dataset with 125,000 blocks, it would be estimated that each storage provider would require the following times to send and retrieve the data:

$$perf_1 = d/r_1 = 125{,}000/.2932 = 426{,}330\text{ms}$$

$$perf_2 = d/r_2 = 125{,}000/.0189 = 6{,}613{,}756ms$$

$$perf_3 = d/r_3 = 125{,}000/.0087 = 14{,}367{,}816ms$$

The results are illustrated in Figure 15.

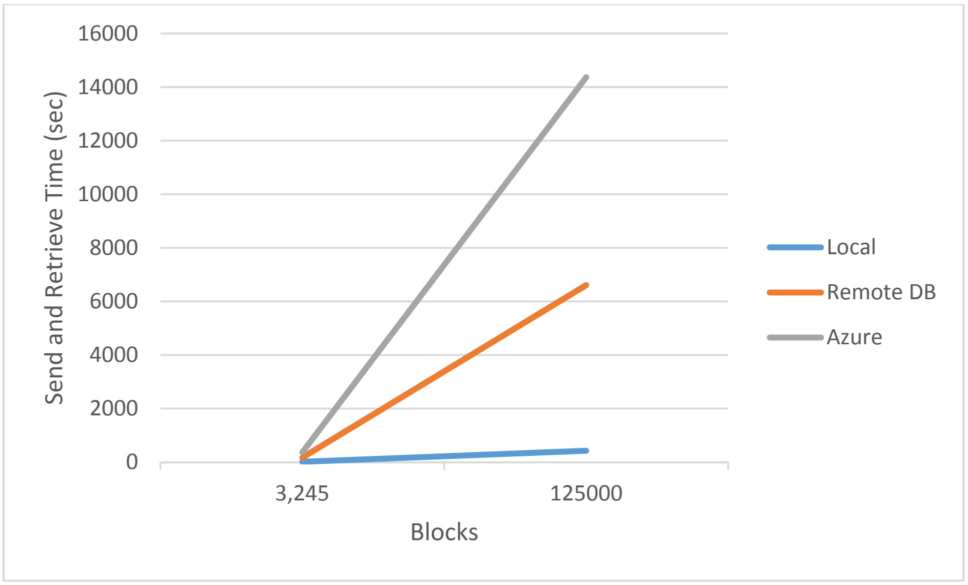


Figure 15. Estimated Provider Performance

The *perf* values as calculated above can now be extended to address a hybrid could scenario by introducing the w_{pub} and w_{pri} values as shown in the equation:

$$\gamma_x = (d_T, r_x, w_x)$$

where γ_x is the time in milliseconds that the Data Key Store to send and retrieve data to and from provider x. In this experiment, w_x signifies the weighted capacity a particular storage provider has in relation to other providers. These values are a percentage of the capacity available in relation to the total storage required, or d_T. This is shown by:

$$w_n = d_x/d_T \text{ when } d_x < d_T$$

$$w_n = 1 - \Sigma(x = 1 \text{ to } n-1)\ w_x \text{ when } d_x > d_T$$

$$w_n = 0 \text{ when } \Sigma(x = 1 \text{ to } n-1) = 1$$

This holds for *n* storage providers where the sum of $w_1 + w_2 + \ldots + w_n = 1$. In order to simulate real world storage limitations, the private cloud storage provider has a limit of 750 blocks imposed. Once this limit is reached no more data will be stored within the private cloud storage provider. Likewise, it is possible that public cloud storage providers have limits, so a limit of 5,000 blocks is imposed on the public cloud remote database storage provider. Finally, it is also possible that a public storage provider has no practical limit to the amount of data that can be stored. This is one of the attractive features of some public cloud offerings. Therefore, the Microsoft Azure Blob storage has a limit of 0, representing no limit. Let w_1 represent the private cloud storage, w_2 represent the public cloud remote database storage and w_3 represent the public cloud Azure Blob storage. In the case of the imposed storage limitations $w_3 = 1 – (w_1 + w_2)$.

Applying these block limits to the data used in the benchmark experiment results in the following

$$w_1 = d_x/d_T = 750/3245 = .2311$$

$$w_2 = d_x/d_T = 1 - .2311 = .7689$$

$$w_3 = 0$$

These values can be used to estimate γ_x values for each provider as follows:

$$\gamma_1 = F(d_T, r_1, w_1) = F(3{,}245, .2932, .2311)$$

$$\gamma_2 = F(d_T, r_2, w_2) = F(3{,}245, .0189, .7689)$$

$$\gamma_3 = F(d_T, r_3, w_3) = F(3{,}245, .0087, 0)$$

The function used to calculate γ_x is defined by:

$$\gamma_x = (\mathrm{d}_\mathrm{T} \text{ x } w_x)/r_x$$

Therefore, the γ_x values for the benchmark experiment are:

$$\gamma_1 = (d_1 \text{ x } w_1)/r_1 = (3{,}245 \text{ x } .2311)/.2932 = 2557.71ms$$

$$\gamma_2 = (d_1 \text{ x } w_1)/r_1 = (3{,}245 \text{ x } .7689)/.0189 = 132014.84ms$$

$$\gamma_3 = (d_1 \text{ x } w_1)/r_1 = (3{,}245 \text{ x } 0)/.0087 = 0ms$$

The results collected from multiple runs with the benchmark data in appendix A. The average actual value for the private cloud storage provider is 2580ms and the average actual value for the public cloud remote database storage is 136322ms. Both of these values are in line with estimates.

This experiment was run with a d_T value of 10,297. The results can be found in appendix A and are summarized in Table 9.

x	w_x	γ_x	**Actual Average**	d_T
1	.2311	2557.71ms	2580ms	3245
2	.7689	132014.84ms	136322ms	3245
3	0	0ms	0	3245
1	.0728	2556.69ms	2333.67ms	10297
2	.4856	264562.07ms	263960.33ms	10297
3	.4416	522661.52ms	519304.00ms	10297

Table 9. Experiment Results

These equations can be used to estimate much larger datasets. For example, this can be used to estimate running time of the 1000 Genomes Project dataset [25] that is 200TB in size. Therefore the d_T value is 200,000,000,000 blocks. The estimation is shown in Table 10.

x	w_x	γ_x	d_T
1	0.00000000375	2557.71ms	200,000,000,000
2	0.00000002500	264,550.26ms	200,000,000,000
3	0.99999997125	22,988,505,086,206.90ms	200,000,000,000

Table 10. 1000 Genomes Project Estimate

With the current artificial limits in place an extremely high amount of burden is placed on the slowest, yet most abundant storage. The total time is estimated at over 728 years with the current configuration. Setting more realistic storage limits of 500GB for the private cloud local storage and 1TB for the public cloud remote database storage still yields over 700 years to store and retrieve the dataset. This shows that more strategy must be taken to improve the storage and retrieval mechanisms beyond simply switching to the next best provider when a storage provider becomes full.

5.6 Analysis

The different storage providers can be shown in a graph relationship. Figure 16 shows the relationship where the s node is the DKS-LISTENER, each Cloud Computing Storage System is shown as P_x node and the t node is the DKS-GET.

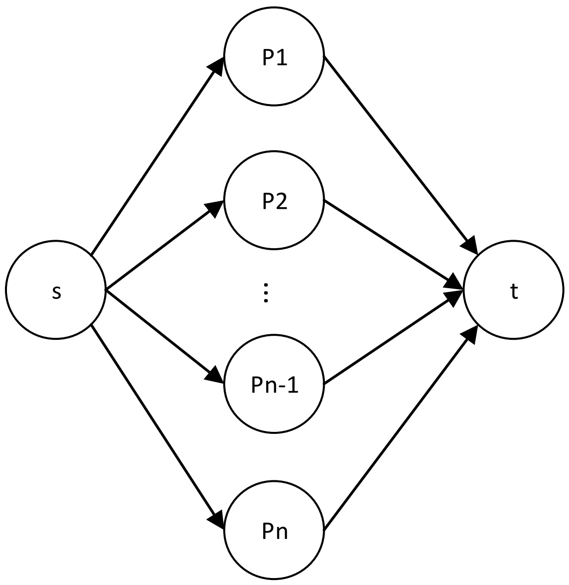


Figure 16. Storage Provider Graph

While this is a simple graph, it allows w_x values to be applied as edge weights. The cost of traveling from s -> P1 -> t is w_1, the cost of s -> P2 -> t is w_2 and so on. If a storage provider becomes full, the weight changes to ∞. Using a greedy strategy when processing data can make a significant difference.

The same performance metrics in previous experiments can be used to show how a greedy algorithm can be used to improve performance. To show this, assume that the Network Traffic Recording System is producing 375 blocks/second and the Hadoop Distributed File system is only capable of processing only 300 blocks/second. This example is in line with the Big Data problem in which data velocity and volume exceed available capacity for storage and processing. Using the current algorithms a total input stream of 9,375 blocks would fill up the private cloud storage in 10 seconds and the public cloud storage would be used to store the remaining data, as shown in Appendix A.

In total, 3,750 blocks are stored in the private cloud local storage and 5,625 blocks are stored in the public cloud storage. By using a greedy algorithm that always retrieves and stores data with the private cloud storage over the public cloud storage it can be shown that 7,950 blocks are stored with the private cloud storage and 1,425 blocks are stored with the public storage provider. Using the calculated performance metrics, the first method is estimated to take 310,408.95ms while the greedy method is estimated to take 102,511.43ms, a significant improvement.

CHAPTER VI

CONCLUSION

Big Data poses a significant challenge to traditional computing models. One of the most significant is challenges is the inability to accommodate the drastic increase in volume and velocity of Big Data. The Industry has looked to cloud solutions to address these challenges, but this poses significant data transfer constraints as proved by this experiment. Introducing the Data Key Store into the Machine Learning Network Topology shows that the system can overcome the increased data volume and velocity associated with Big Data while maintaining the security of the data.

However, this is not enough. As shown in these experiments this leads to another problem. The amount of time spent transferring data quickly becomes problematic as it extends storage time to an unacceptable amount of time. The example of the 1000 Genomes Project dataset [25] shows that the time required is over 700 years which illustrates this point. However, if a greedy strategy is employed, progress towards these limitations can be made. In the example data calculated the data can be processed while making the best use of the private cloud storage, reducing overall runtime by roughly 66%.

In addition to allowing the Machine Learning Network Topology to handle Big Data, the Data Key Store also provides a foundation for necessary operations while analyzing Big Data. The DKS-VERIFY algorithm shows a simple method for verification of data. This provides a method to prove data exists with a cloud provider without having to retrieve large amounts of data.

Areas of further research into the use of hybrid cloud storage center around how to use storage more effectively. As shown in this research ensuring the highest performing storage is used whenever possible is paramount. Other options to improve this should be explored, including applying traditional disk concepts to cloud storage. Possible improvements may be found by using more storage providers. Data resiliency may also be achieved.

In conclusion, it is possible to extend Machine Learning systems to incorporate hybrid cloud storage to meet the increasing amounts of data being produced needing analysis. In addition, this can be accomplished securely without relying on data tagging or limiting the location that specific data can be stored. This overcomes current challenges posed when using hybrid cloud storage with Big Data.

REFERENCES

[1] How Much Information? (2013, July) berkely.edu. [Online]. http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/execsum.htm

[2] IBM. (2012, April) ibm.com. [Online]. http://www-01.ibm.com/software/data/bigdata/

[3] What Twitter Learns from All Those Tweets. (2013, August) MIT Technology Review. [Online]. http://www.technologyreview.com/view/420968/what-twitter-learns-from-all-those-tweets/

[4] Cisco Visual Networking Index. (2013, August) cisco.com. [Online]. http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.html

[5] High Performance Computing (HPC) on AWS. (2013, March) Amazon Web Services. [Online]. http://aws.amazon.com/hpc-applications/

[6] P.C. Zikopoulos, C. Eaton, D. deRoos, T. Deutsch, and G. Lapis, Understanding big data – Analytics for enterprise class Hadoop and streaming data, McGraw-Hill, 2012.

[7] About Twitter. (2012, March) twitter.com. [Online]. https://about.twitter.com

[8] Welcome to Apache Hadoop! (2013, August) The Apache Software Foundation. [Online]. http://hadoop.apache.org

[9] S. Suthaharan, “Big Data Classification: Problems and challenges in network intrusion prediction with machine learning,” Big Data Analytics workshop, in conjunction with ACM Sigmetrics 2013 (under review).

[10] Big Data. (2013, August) Gartner. [Online]. http://www.gartner.com/it-glossary/big-data/

[11] Boyd, Danah and Crawford, Kate, Six Provocations for Big Data (September 21, 2011). A Decade in Internet Time: Symposium on the Dynamics of the Internet and Society, September 2011.

[12] Moore’s law and how cloud has changed Moore’s law

[13] Peter Mell and Timothy Grance. (2011, September) The NIST Definition of Cloud Computing. Document.

[14] Microsoft. (2012, April) microsoft.com. [Online]. http://www.windowsazure.com/en-us/home/features/compute/

[15] Amazon Web Services. (2012, April) Amazon Web Services. [Online]. http://aws.amazon.com/ec2/

[16] J. Lin and M. Schatz. 2010. “Design patterns for efficient graph algorithms in MapReduce.” In *Proceedings of the Eighth Workshop on Mining and Learning with*

Graphs (MLG ’12). ACM, New York, NY, USA. 78-85. DOI=10.1145/1830252.1830263 http://dx.doi.org/10.1145/1830252.1830263

[17] H. Herodotou and S. Babu. 2011. Profiling, What-if Analysis, and Cost-based Optimization of MapReduce Programs. In Proc. of the 37th International Conference on Very Large Data Bases (VLDB '11). 1111-1122 http://www.vldb.org/pvldb/vol4/p1111-herodotou.pdf

[18] Life Sciences. (2013, October) Amazon Web Services. [Online]. http://aws.amazon.com/lifesciences/

[19] DNA Sequencing Costs. (2013, October) National Human Genome Research Institute. [Online]. http://www.genome.gov/sequencingcosts/

[20] Kehuan Zhang, Xiaoyong Zhou, Yangyi Chen, XiaoFeng Wang, and Yaoping Ruan. 2011. Sedic: privacy-aware data intensive computing on hybrid clouds. In *Proceedings of the 18th ACM conference on Computer and communications security* (CCS '11). ACM, New York, NY, USA, 515-526. DOI=10.1145/2046707.2046767 http://doi.acm.org/10.1145/2046707.2046767

[21] A. Juels and B. S. Kaliski Jr, PORs: Proofs of retrievability for large files. In *Proceedings of the 14th ACM conference on Computer and communications security* (CCS ’07). ACM, New York, NY, USA, 584-597. DOI=10.1145/1315245.1315317 http://doi.acm.org/10.1145/1315245.1315317

[22] H2 Database Engine. (2013, August) h2database.com. [Online].
http://www.h2database.com/html/main.html

[23] MySQL Community Edition. (2013, July) MySQL. [Online].
http://www.mysql.com/products/community/

[24] Les Miserables by Victor Hugo. (2013, March) Project Gutenberg. [Online].
http://www.gutenberg.org/ebooks/135

[25] 1000 Genomes Project. (2013, November) Amazon Web Services. [Online].
http://aws.amazon.com/datasets/4383

APPENDIX A

EXPERIMENT RESULTS

Benchmark Experiment with 3,245 total blocks.

Provider	eType	SEND Elapsed Time(ms)	GET Elapsed Time (ms)	GET DECRYPT (ms)
Azure Blob	1	247679	123807	0
Azure Blob	1	251256	122432	0
Azure Blob	1	252060	123216	1
Azure Blob	2	256216	121442	475
Azure Blob	2	255166	121547	475
Azure Blob	2	254478	121175	496
Azure Blob	3	248835	121164	317
Azure Blob	3	248177	121405	323
Azure Blob	3	248290	121588	367
Local	1	5680	5864	1
Local	1	5066	6289	1
Local	1	4612	5689	1
Local	2	4394	5785	600
Local	2	4354	5543	600
Local	2	4605	5396	598
Local	3	4893	5659	452
Local	3	4718	6247	440
Local	3	4414	6571	435
Remote DB	1	111764	68729	0
Remote DB	1	112039	55191	1
Remote DB	1	111534	55268	0
Remote DB	2	113203	54474	575
Remote DB	2	113958	54614	576
Remote DB	2	113122	54555	576
Remote DB	3	114985	68257	423
Remote DB	3	113145	55588	420
Remote DB	3	115091	55173	419

Experiments with 3,245 and 10,297 blocks and artificially imposed limits to storage providers.

Run	Input Size	Provider	eType	Max Block	SEND Elapsed Time (ms)	GET Elapsed Time (ms)
1	3245	Local	1	750	1634	1292
1	3245	Remote DB	2	5000	92155	41650
1	3245	Azure Blob	3	0	0	0
2	3245	Local	1	750	1309	1174
2	3245	Remote DB	2	5000	91609	50331
2	3245	Azure Blob	3	0	0	0
3	3245	Local	1	750	1239	1092
3	3245	Remote DB	2	5000	91256	41966
3	3245	Azure Blob	3	0	0	
4	10297	Local	1	750	1129	1112
4	10297	Remote DB	2	5000	173967	102459
4	10297	Azure Blob	3	0	349134	168850
5	10297	Local	1	750	1205	1110
5	10297	Remote DB	2	5000	176654	83006
5	10297	Azure Blob	3	0	349134	171415
6	10297	Local	1	750	1339	1106
6	10297	Remote DB	2	5000	172796	82999
6	10297	Azure Blob	3	0	348831	170548

Regular algorithm processing 32 blocks of input size 375 and capability to process 300 blocks/second.

Second	Stored in Pri	Retrieved from Pri	Capacity of Pri	Stored in Pub	Retrieved from Pub	Capacity of Pub
1	375	300	75	0	0	0
2	375	300	150	0	0	0
3	375	300	225	0	0	0
4	375	300	300	0	0	0
5	375	300	375	0	0	0
6	375	300	450	0	0	0
7	375	300	525	0	0	0
8	375	300	600	0	0	0
9	375	300	675	0	0	0
10	**375**	**300**	**750**	0	0	0
11	0	0	0	375	300	75
12	0	0	0	375	300	150
13	0	0	0	375	300	225
14	0	0	0	375	300	300
15	0	0	0	375	300	375
16	0	0	0	375	300	450
17	0	0	0	375	300	525
18	0	0	0	375	300	600
19	0	0	0	375	300	675
20	0	0	0	375	300	750
21	0	0	0	375	300	825
22	0	0	0	375	300	900
23	0	0	0	375	300	975
24	0	0	0	375	300	1050
25	0	0	750	375	300	1125
26	0	300	450	0	0	1125
27	0	300	150	0	0	1125
28	0	150	0	0	150	975
29	0	0	0	0	300	675
30	0	0	0	0	300	375
31	0	0	0	0	300	75
32	0	0	0	0	75	0
	3750	**3750**		**5625**	**5625**	

Greedy algorithm processing 32 blocks of input size 375 and capability to process 300 blocks/second.

Seconds	Stored in Pri	Retrieved from Pri	Capacity of Pri	Stored in Pub	Retrieved from Pub	Capacity of Pub
1	375	300	75	0	0	0
2	375	300	150	0	0	0
3	375	300	225	0	0	0
4	375	300	300	0	0	0
5	375	300	375	0	0	0
6	375	300	450	0	0	0
7	375	300	525	0	0	0
8	375	300	600	0	0	0
9	375	300	675	0	0	0
10	75	300	450	300	0	300
11	200	300	350	175	0	475
12	375	300	425	0	0	475
13	325	300	450	50	0	525
14	300	300	450	75	0	600
15	300	300	450	75	0	675
16	300	300	450	75	0	750
17	300	300	450	75	0	825
18	300	300	450	75	0	900
19	300	300	450	75	0	975
20	300	300	450	75	0	1050
21	300	300	450	75	0	1125
22	300	300	450	75	0	1200
23	300	300	450	75	0	1275
24	300	300	450	75	0	1350
25	300	300	450	75	0	1425
26	0	300	150	0	0	1425
27	0	150	0	0	150	1275
28	0	0	0	0	300	975
29	0	0	0	0	300	675
30	0	0	0	0	300	375
31	0	0	0	0	300	75
32	0	0	0	0	75	0
	7950	**7950**		**1425**	**1425**	