ABSTRACT

Migrating Legacy Software Applications to Cloud Computing Environments: A Software Architect's Approach

by Fred Rowe

December, 2011

Director of Thesis: Dr. M. Nassehzadeh Tabrizi

Major Department: Computer Science

Cloud computing has garnered a great deal of interest in the past few years. The availability of on-demand computational power is presumed to provide substantial IT infrastructure cost-savings, partially through the reduction of maintenance and administration costs. However, in order to take advantage of these savings, it is often required that legacy applications be rewritten at least partially, if not in entirety, to operate in these environments. As a part of re-architecting these legacy assets for cloud computing environments, the software architect may also consider application modifications providing other cost benefits which may have been cost prohibitive to implement in a more traditional computing environment. Although not a new technology, the combination of parallel computing and cloud environments can offer a number of benefits to many application categories if the cost of making the necessary changes to the application and setting up and maintaining the environment can be justified.

This thesis explores the use of cloud computing to provide a flexible deployment environment in which to run a migrated legacy application using one of the popular parallel computing frameworks. The ability to easily and rapidly configure and deploy hardware and software to create a cloud capable of executing applications with parallelism combines the

benefits of these technologies in a powerful manner. In order to make an informed decision about the potential benefits of such an environment, the owner of those assets needs to be able to balance any savings against any costs incurred to enable existing corporate business applications to run in such an environment.

An approach to performing such an analysis is presented in this thesis. To provide some quantitative means of measuring benefits, benchmark results of the computational resources required by the application in the different environments are provided. Additionally, offsetting costs such as software re-architecting and refactoring are considered.

Migrating Legacy Software Applications to Cloud Computing Environments: A Software Architect's Approach

A Thesis

Presented To the Faculty of the Department of Computer Science

East Carolina University

In Partial Fulfillment of the Requirements for the Degree

Master of Science in Software Engineering

by

Fred Rowe

December 2011

Migrating Legacy Software Applications to Cloud Computing Environments: A Software Architect's Approach

by

Fred Rowe

APPROVED BY:

DIRECTOR OF THESIS: _____

M. H. Nassehzadeh Tabrizi, PhD

COMMITTEE MEMBER: _____

Karl Abrahamson, PhD

COMMITTEE MEMBER:_____

John Placer, PhD

COMMITTEE MEMBER:_____

Sergiy Vilkomir, PhD

CHAIR OF THE DEPARTMENT OF COMPUTER SCIENCE:

_____

Karl Abrahamson, PhD

DEAN OF THE GRADUATE SCHOOL:

_____

Paul J. Gemperline, PhD

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

There is a great deal of interest and investment in the emerging technology of cloud computing. One of the many touted benefits of cloud computing is the availability of on-demand computational power which can be consumed on an "as-needed" or "pay-as-you-go" basis [1]. There is the presumption that the cost-savings of not having to maintain and administer the computing infrastructure will more than offset the cost incurred (whether it is an internal or external charge) when paying only for the cloud resources utilized. However, to compare the true costs, any savings from an infrastructure perspective must be offset with any costs associated with the migration of a company's legacy enterprise applications to the new platforms. It is often the role of the software architect to perform a comparative analysis which validates any benefits from such a migration [2].

The term legacy application refers to an enterprise or business application, often written in a older programming language like FORTRAN or COBOL, for which businesses rely on to perform routine daily Information Technology (IT) tasks like accounting and payroll. These applications may also be less mundane and instead perform specialized and proprietary business processes, like text mining transactions for patterns and Business Intelligence (BI). The software architects of such businesses are constantly challenged to keep these assets running at the lowest cost. This includes challenges such as maintenance of code for which programmers may not be readily available and for which the source has become a jumble of defect repairs making even minor modifications costly in terms of risk and resources. In addition to these challenges, companies are tightening IT infrastructure budgets and thus architects are looking for ways to run these back-office applications less expensively without investing in new IT infrastructure.

Because of all the attention and press cloud technology is receiving, it has likely come to the software architect's attention that the latter challenge might be met by this technology.

The intent of this thesis is to examine the thought process and stages a software architect might progress through in analyzing the possibility of exploiting the new cloud architectures to reduce the cost of operation of these legacy applications. In order to do so, we will follow the migration of a text-mining application, acting a proxy for any legacy application, through a set of stages progressing towards deployment in a cloud environment. In addition to examining cloud environments, because the software architect should consider as many technical solutions as possible, the use of parallel processing in a cloud to perform the text mining is examined. A software architect might reasonably expect that the ease of deploying repeatable patterns of infrastructure in a cloud could overcome some of the costs typically associated with parallel computing like configuration, maintenance and administration of the required hardware and software while reducing the execution time and thus cost of complex text mining operations.

This chapter (Chapter 1) provides an introduction into the research behind this thesis including the motivation of a software architect to consider the migration of legacy applications, the rationale behind doing so, and some of the thought process involved in planning such a project. Chapter 2 provides a high level overview of some of the facets of cloud computing which are pertinent to this research. This background material includes a discussion of cloud service layers, the classification of private, hybrid and public clouds and those aspects of parallel computing in a cloud environment which are important to this research. The background continues with an analysis of the need to justify any benefits resulting from using both parallel computing and cloud technologies and why those benefits are important to the software architect. Issues specific to this research such as dealing with large data files and the effects on tradeoffs

which the architect must make are then discussed. Next, the constraints of the environment in which the software architect is working and its potential impact on the architect's decisions are presented. Finally, the role of the software architect is explained in order to help those not familiar with the position to better understand how their interaction with the various other team members affects the outcome of the project.

Chapter 3 is a literature survey of some of the published work which closely relates to this research and helps to provide context to areas of common issues. Chapter 4 describes the means and methods that I used to carry out this research. The chapter includes a discussion of the hardware and software environment used to perform the benchmarking, the iterations used to develop and test the applications, and the various settings of the deployed environment which were required to configure the parallel computing cloud. Finally, Chapter 5 presents the benchmark results in graphical and tabular form, while Chapter 6 provides a discussion of what the results mean, how the software architect might assess the results, and a comparison of some of both the quantifiable and intangible benefits which might be gained from completing a project similar to this research.

Rather than create a separate section for motivation as is sometimes done in theses, I have interjected my motivation and rationale throughout my thesis as appropriate, in order to better provide context for the decisions and factors driving them.

CHAPTER 2: BACKGROUND

Like any new technology, one of the challenges facing the software architect is the learning curve of first understanding the technology and then the more difficult task of appraising how, if at all, that technology might be useful to the problem space for which the architect is responsible. Because of coverage given to cloud computing and related topics by the scientific press, fortunately the software architect will find a number of readily available resources to help with the first task [3-5]. The second task is one which challenges the skill and experience of the architect and will depend on new approaches to thinking about a problem and the associated requirements [2].

2.1 Cloud Computing

Cloud computing is not a concept easily defined in specific terms because it has been applied to so many architectures and technologies. From the National Institute of Standards and Technology (NIST), Mell and Grance [6] defined it broadly as follows:

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

2.1.1 Cloud Service Layers

The above definition of cloud computing hints at the many facets of cloud architectures including storage, applications and platforms without providing specifics on any of them. Because every component of modern IT infrastructure has been incorporated (sometimes in

multiple ways) into a cloud, it is typical to define several broad classes of services offered by cloud providers: Infrastructure as a Service (IaaS), Software as a Service (SaaS) and Platform as a Service (PaaS) [3]. Because this thesis examines issues related to the migration of legacy applications and the hosting of such an application in a cloud capable of parallel computing, it will focus on a form of PaaS. Those layers of service are visualized in Figure 1.

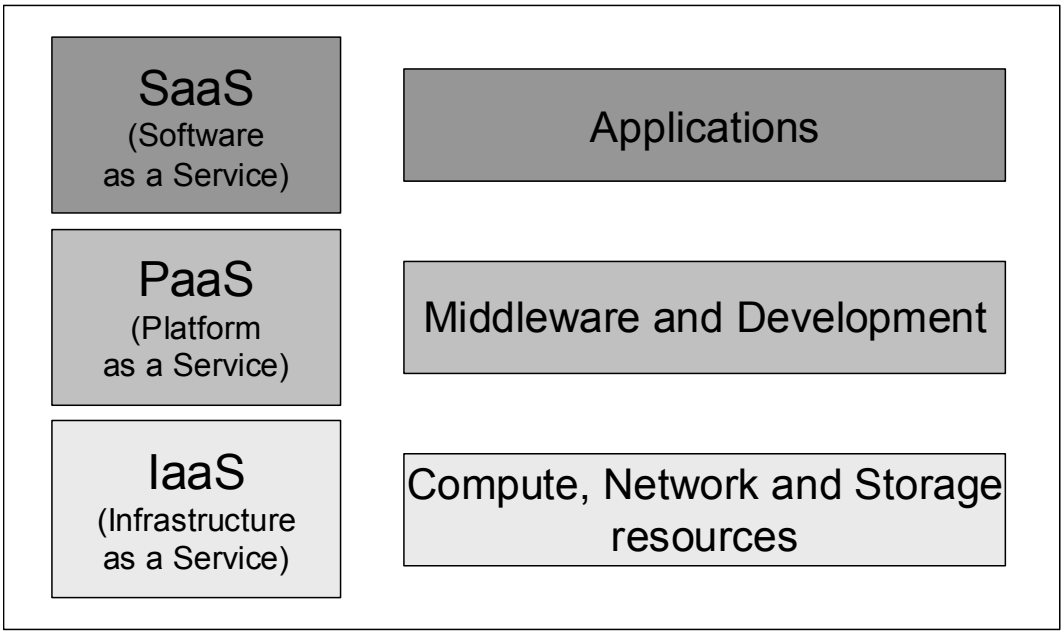

Figure 1: Typical layering of cloud service offerings

IaaS is built upon a virtualization layer of one or more virtual machines and provides the hardware and software which allow systems administrators and developers to provision processing, storage, network and other resources needed to deploy and run their Operating Systems (OS) and applications. Built on top of, and leveraging the services of IaaS, PaaS is an environment supporting both software development and application hosting, and is composed of a set of development tools, databases, middleware, and infrastructure software. The set of tools offered by the PaaS provider tends to be vendor-specific and typically leverages their software stack. For example, if an application requires the parallel computing capabilities of a

MapReduce implementation and the usage of Open Source Software (OSS) is desirable to the deployer; one or more Hadoop nodes might be provisionable using the PaaS tools. Because a PaaS cloud is built on top of the services provided by the lower layers and is thus insulated from them, the software architect is able to focus more the details of application composition and less on the IT environment [7].

2.1.2 Private, Public and Hybrid Clouds

In addition to the classification by service offering layers, clouds are typically further categorized by the ownership of the resources comprising the cloud. A *private cloud* is one in which the software and hardware components are owned and used by an organization for its internal purposes. These resources are typically within corporate firewalls and may be owned at the organization level and made available on-demand to various departments across the organization or may be owned by smaller entities within the organization for specific needs and only available to those entities. If provided by another department or branch of the organization, there may be a cost associated with the resources which may accounted for by internal billing. As an example, a functional test department of a software development organization may decide to consolidate and redeploy it's existing IT assets in a private cloud for use only by that specific department, limiting access to authorized testers.

At the opposite end of the spectrum are *public clouds*, easily accessible via the Internet and offered for public consumption by a provider like Amazon's EC2 (Elastic Compute Cloud) [8] or IBM's SMART cloud [9]. Each provider has their own payment model with many offering free accounts, albeit limited by either time or resources, to encourage prototyping and sampling of the services by architects, developers and the like. *Hybrid clouds* are composed of resources from

private and public cloud offerings. Deployments of hybrid services are predicted to grow, particularly in some specific usage patterns, like accessing data warehoused in private corporate data centers using public cloud software stacks [1].

2.1.3 Parallel Computing in a Cloud with Hadoop

Having considered some of the cloud service options available, a software architect might also want to consider, as discussed earlier, the ability to execute a legacy application in a parallel computing environment deployed in a cloud. There are a number of reasons this capability might be an attractive feature of the new architecture for which the architect is developing plans. One advantage might be a reduction in either computing resources or time to perform routine data processing. It is reasonable for the architect to ask for example, if text mining (or some other processing) is routinely performed on a dataset, could the execution time of that execution be reduced by migrating the legacy application to a parallel processing environment? Prior to the advent of cloud computing, even if such an experiment proved to be cost-effective, the cost of deploying and maintaining the additional (and probably dedicated) infrastructure would have likely offset any savings. However, the combination of a cloud with the appropriate parallel processing environment installed and configured, deployed on-demand and without intervention could be very cost-effective for the scheduled execution of a data mining run. In this architecture, the computing resources could be consumed only during the scheduled execution and then released back to the pool of hardware and software for use by other applications providing for significantly lower administration and operation costs shared amongst a pool of cloud users.

So the architect may be able to justify consideration of parallel computing clouds in their plans, but just as there is a learning curve associated with cloud technology, the software architect will have to invest at least enough time to understand the basics of parallel computing and how to most effectively integrate such an environment with any legacy assets. As a part of that learning and evaluation curve, the architect may find it helpful to prototype several phases of the project being contemplated in order to validate the design. While prototyping a new design, an architect will often investigate any available OSS implementations of a new technology which they are interested in incorporating in the design. Because the architect can typically acquire the OSS at no cost, it is often easier to justify the prototyping and evaluation phase of the project to the stakeholders. For these reasons, I chose to use Hadoop (an OSS implementation of Google MapReduce) in this research. Like cloud computing, there are numerous resources available to assist in educating the architect on the Hadoop software [10], [11] and [12].

2.1.4 Hadoop Operational Modes

Having decided to incorporate Hadoop into the envisioned architecture, the architect would need some understanding of the environments in which Hadoop can be run. There are three different modes for which it may be configured [15]:

- Standalone mode – also known as local mode, there are no daemons running and everything runs within a single JVM. This is the default mode of operation and is suitable for running MapReduce programs during development, since it is easy to test and debug them. In this mode, a single Hadoop node is created but does not use the Hadoop Distributed File System (HDFS). This means that all input and output files are read from/written to the underlying OS file system, thus there will be no benefits from using HDFS.

- Pseudo-distributed mode – all of the Hadoop daemons run on the local machine, thus simulating a cluster on a single machine. Other than where the Hadoop processes are running (one machine vs. one machine per node), this mode is the same as distributed mode. In this mode, Hadoop starts all the processes for all the configured nodes on the same machine. This mode is useful because it allows the architect to observe how applications respond to running on a Hadoop cluster, but without the overhead of setting up the individual machines for the nodes of the cluster. While that task is much easier for a software architect using a cloud and a Hadoop VMI, there is still some overhead involved. Because the HDFS is used by default, benefits may be gained from using it.
- Fully distributed mode –the Hadoop daemons run on a cluster of machines. Each Hadoop node is started on the specified machine. As with pseudo-distributed, HDFS is used.

The importance and implications of these options will be discussed later in the Methodology section.

2.2 Quantifying Benefits

As discussed previously, analysis of the costs and benefits of a software project often falls to the software architect. How to quantify any benefits from migrating a legacy application from a traditional computing environment into a parallel computing cloud? One method would be to compare the overall execution time of the legacy application executing in a cloud with the execution time required on a more traditional stand-alone hardware and software platform. Although this is a less sophisticated means of comparison, the software architect is often interested in performing a "sniff test"; a coarse-grained test to determine whether the design

under consideration merits further iteration or needs to be completely revised. For the purpose of this thesis, the benchmarks will compare the relative execution time of the three versions of an application in various configurations, looking for improvements that signal the architecture has merit and should be further explored.

2.3 Big Data

In performing this research, it became clear that one of the challenges of current text-mining applications is the storage, handling and processing of massive amounts of data. The term big data is commonly used to describe the petabytes of data which are produced by the typical organization and which must be processed in order to provide the business intelligence to drive all manner of marketing, sales, logistics, service and other activities of the organization. The software architect of such an application would be acutely aware of many of the issues with handling such large volumes of data including transfer, storage and processing of multi-megabyte files. To that end, the software architect, in considering options for migrating such an application would want to evaluate the capabilities of the HDFS which was designed specifically for handling massive datasets [12]. This capability is another reason that Hadoop was selected for this research.

2.4 Architectural Constraints

As the software architect considers the possibility of utilizing cloud resources, what is available to them will often depend on the infrastructure provided by the corporate IT office. In the best case scenario, corporate would be able to provide cloud facilities to the architect on which to execute some prototyping and test scenarios to help begin the comparative benchmarking of various scenarios. In the worst case, corporate IT might do little more than

provide some hardware preloaded with an operating system to the architect. For the purpose of this thesis, I examine a middle road, where IT is capable of providing a private cloud which the architect can use to perform an analysis using the tools best suited to the project.

Although both Cloudera and Amazon Web Services (AWS) have commercial PaaS offerings for Hadoop, for several reasons I choose instead to use a minimal toolset for this research and to create a private PaaS cloud instead. One reason relates to cost; if the software architect is trying to justify the expense of migration, it may be easier to do so using a private cloud from corporate IT. The second reason is related to the very large datasets processed by the text-mining application. From a project standpoint, it is easier for the architect to avoid addressing the logistics and cost of accessing those large datasets on a public cloud, at least during the early, prototyping phases of the project, so a private PaaS cloud made more sense.

2.5 The Software Architect and the Project Stakeholders

To this point, I've used the term software architect a great deal but without providing a definition of the role, partly because the term is used to describe a broad range of job responsibilities and often has a different meaning at different organizations. McBride gave a good overview of the role, saying: "Fundamentally different ways of thinking about design and interacting with systems and stakeholders represent the essence of the software architect [2]." For the purpose of this thesis, the software architect is a technical person, often an engineer, responsible for the development and maintenance of a software application, or a framework or set of applications. As the person responsible for the lifecycle of that application, the architect will often need to investigate new technologies for potential benefits and make a

recommendation to the stakeholders as to whether to incorporate that technology into the application. A pattern for executing such an evaluation is outlined in this thesis.

In addition to the architect, many other roles within the software organization will typically be stakeholders in the application and thus affected by any proposed changes to it. From the development organization, software *developers* who are responsible for the development and maintenance will need to be convinced by the evaluation that the work to migrate the application will not require completely rewriting the application. From the test organization, software testers will need reassurance that the migration and new environment will not introduce new defects in the application. *Product management* will need to understand what long-term risks of the new architecture are, and that these risks can be managed and are outweighed by the benefits. Finally, the *executive sponsor* will need to be convinced that the investment is justified. The evaluation performed by the architect and the quantification of the measured benefits should be used in conjunction with additional project plans to address the concerns of each of these stakeholders.

CHAPTER 3: RELATED WORK

McBride [2] provides significant insight into the role of the software architect, the traits that are necessary to be successful as one, and most important to this research, examples and characteristics of the thought processes followed by architects in the execution of successful projects. Because much of the research described in this thesis proceeds in a logical progression based on the steps which a software architect could use to try to quantify specific benefits and costs of a migration project, it is useful to understand from other project perspectives how that process evolves. One has to be careful in reading McBride's work however, because at a causal glance, he would seem to say that iterative development is "a disaster" which would be antithetical to the approach taken by my research. A closer examination reveals that he is instead indicting the "common usage" of evolutionary design which is too often unrestrained and undisciplined and not driven by a rational process.

Prodan [7] provides a useful system of categorizing and classifying the type of clouds which are typically encountered in practice. His assertion that PaaS is "relatively new and immature" is indication of just how fast cloud technology is moving because of the number of mature PaaS offerings like Google App Engine and Microsoft Azure which are now available only two years later. In spite of that, his work provides a useful survey of some of the differentiating characteristics of the available clouds systems which can help the software architect better understand which of those features might be useful.

Louridas [3] compares grid and cloud computing; for those software architects familiar with grid computing, the similarities and differences will prove useful. His overview of putting

together a private cloud meshes well with the steps laid out in my research. Although his work provides a discussion of cloud support for parallelism, it is too brief to be of use to my research.

The work of Tran et al [13] correlates well with my research. Both of our research describes the migration of applications to PaaS environments, but whereas they use commercial cloud offerings (Azure and Amazon EC2); I chose to implement a minimal PaaS environment targeting only those tools I needed. Although their work focuses on the migration of an entirely different category of application (enterprise applications based on either .Net or Java Enterprise Edition), the taxonomy of tasks required for migration is useful. Their observation that the time spent to setup and configure the new environment contributes significantly to the overhead cost of the migration is justification of the central concepts of this thesis, that the ability to setup and configure the environment once and then to be able to easily reproduce that environment on demand then spreads the cost of that overhead over the life of the application. Similar to my research, their approach towards justification of the decision to migrate an application is based on a methodical process.

CHAPTER 4: METHODOLOGY

In designing the research for this thesis, I followed the thought process a software architect might use in designing a project with the goal of determining whether there were cost benefits in migrating a legacy application to new environments.

4.1 Application Deployment and Execution Environments

As a part of designing a project to quantify these benefits, the architect would need to establish the configurations in which the versions of the application would be benchmarked. Because a baseline benchmark is needed to which to compare the other results, a computing environment suitable for executing the legacy Java application is needed. Then, because the architect intends to experiment with the migration of that application to one or more versions capable of running as a Hadoop application, a computing environment suitable for executing Hadoop applications is needed. These two environments were chosen to represent a path for migrating an application from a typical enterprise application execution environment (a Java application) to a cloud environment capable of parallel processing (a Hadoop version of the same application).

As discussed previously in the Constraints section, this research will utilize resources from a private cloud comprised of commodity-class hardware. Onto this hardware the architect would deploy virtual machine images (VMI) which were created using the tools associated with the particular cloud infrastructure provided by the IT department. These images consist of the software stacks in which the applications are executed. A software stack is a set of software components layered upon each other, interacting through well-defined (often by a standards body) application and system programming interfaces (API and SPI) to provide the services

required for the deployment and execution of applications. For this research two VMI were created. The first VMI supports deployment of Java applications requiring only the Java Standard Edition (JSE) Java Runtime Environment (JRE); the second supports Hadoop applications requiring both the JSE JRE and the Hadoop system. A further description of the two VMI will help one to understand the role they play in my research.

4.1.1 The Java Runtime Environment Virtual Machine Image

The JRE VMI was chosen as the first step towards migrating from a traditional computing platform into a cloud environment. In this configuration, a VMI was created consisting of the Linux OS and a JSE JRE per the following specifications:

- OS: SUSE Linux version 2.6.32.27-0.2-pae
- JSE JRE: IBM J9 VM (build 2.6, JRE 1.6.0 Linux x86-32 20110322_78375 (JIT enabled, AOT enabled)

As depicted in Figure 2 this environment provides all the necessary components to deploy and execute a standalone Java application. When deployed to a cloud node, this VMI creates the appropriate PaaS environment comparable to a single processor computer with hard disk storage and with a version of the Linux operating system and a JSE JRE installed. A Java application which could be executed on such a platform should be able to run without modification within this VMI. The software architect might choose this configuration as the first step because of the similarity to the environment in which the legacy application is already running.

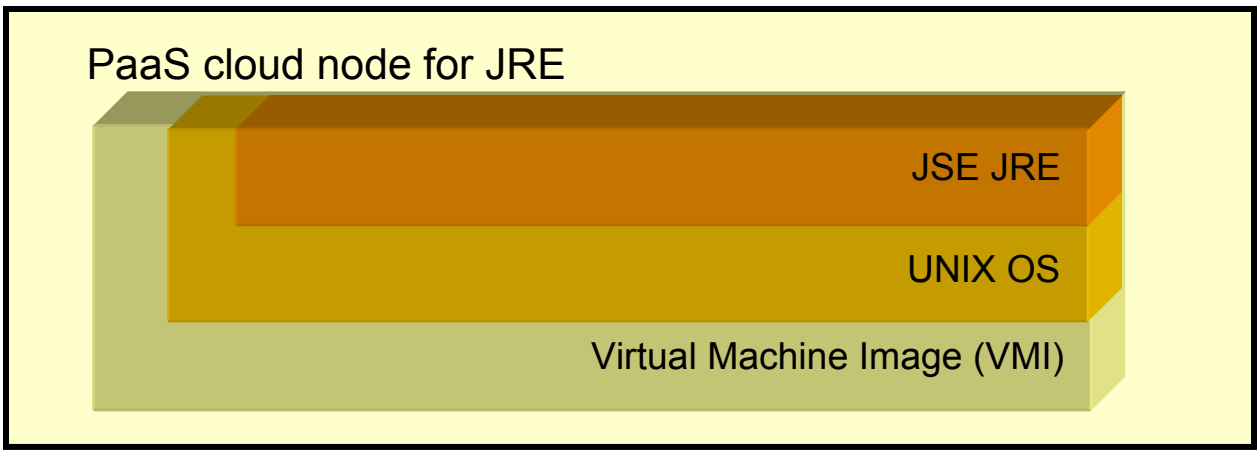

Figure 2: A PaaS cloud node suitable for running JSE Java applications

4.1.2 The Hadoop Virtual Machine Image

In this configuration, the JRE VMI was extended by installing the Hadoop version 0.20.2 distribution into the image. As depicted in Figure 3 this environment provides all the necessary components to deploy and execute the Hadoop versions of the application. When deployed to a cloud node, each instance of the VMI creates the appropriate PaaS environment comparable to a single processor computer with hard disk storage, and with a version of Linux, a JRE and the Hadoop software installed.

In spring 2011 when the research for this thesis was performed, the 0.20.2 release of Hadoop was the latest stable (bug-fixes only) version. My original intent was to use the latest release from Apache (which is 0.21 at this writing), but JIRA bug HADOOP-6941 prevented that release from running on an IBM JRE (or any non-Sun JRE). This incompatibility wasn't discovered until after the benchmarking for the standalone Java application had been completed. Rather than switch the JRE and potentially introduce another variable in the comparison between

environments, the earlier release was chosen for its compatibility with the IBM 1.6 JRE.

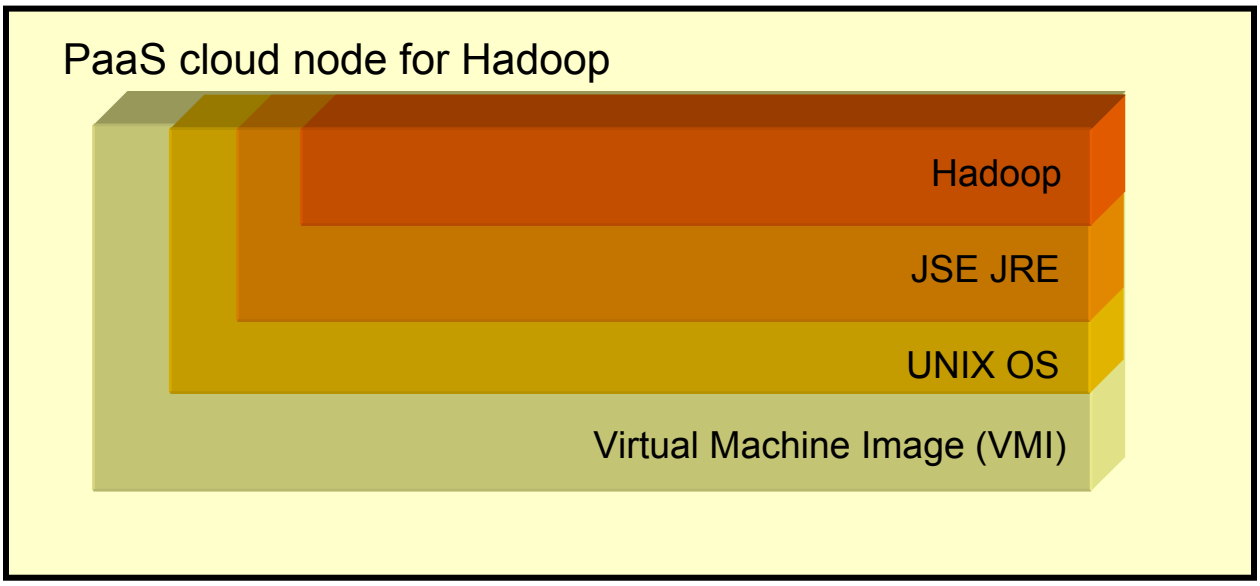

Figure 3: A PaaS cloud node suitable for running Hadoop applications

An architect is often faced with similar conflicts, particularly as an early-adopter of open source software (OSS). To further the discussion started in the Constraints section, many corporate IT organizations have strict limitations not only on the platforms, operating systems, Java runtime environments, etc which they allow to be placed on the corporate network, but also may often have restrictions related to specific versions, releases, patch levels, etc. Having to use a certified version may limit the architect's choices with respect to software packages. In this project, because there was no functionality in the newer release which was required to run the application, the issue was of no real consequence, however in the real world, the latest release is often more desirable because of specific functionality or as a minimum baseline on which to base the migration of the application. As always, whether with OSS or Commercial Off-The Shelf (COTS) software, the architect must also balance the stability of a less-than latest release with the potential feature set of the latest release.

4.1.3 Hadoop Operational Modes, File System Block Sizes, and Clusters

Recall from the Background section that Hadoop can be configured in three different modes:

- Standalone
- Pseudo-distributed
- Fully distributed

In looking at these modes and the increasing complexity thereof, it seems a natural progression for the architect to start working with Hadoop in standalone mode and as experience is gained, progress next to pseudo-distributed and then finally to distributed mode. This progression allows the architect to invest minimally in up-front education, configuration and setup, and instead focus on experimenting with the software and learning by usage.

In addition to configuring the mode of operation, this research also examined the effects of adjusting the HDFS block size. HDFS blocks are the basic unit of storage management, and the basis for tracking, allocating, and transferring file storage across nodes. HDFS divides files into blocks based on the block size setting. This size is the amount of space allocated for each replica of the block. Because tuning this parameter can have an impact on the performance of the cluster [15] the architect would want to determine the impact of changing the value of this setting on the benchmarks.

Finally, aspects of the scalability of Hadoop were examined. As with many parallel processing systems, one means of achieving scalability with Hadoop is through the use of clustering. In this context, clustering refers to a logical grouping of individual machines cooperating to perform as a single unit. Thus, the software architect would want to experiment with cluster configurations to see what affect they can have on the execution time. As mentioned

previously, the job of the architect is made somewhat easier because of the decision to create a Hadoop VMI which can quickly and easily be deployed to create multiple Hadoop nodes. As was done for this research, the node configuration files would need to be edited to properly setup the cluster once the Hadoop VMI was deployed. For this research, two cluster configurations were chosen for execution of the benchmark tests; a two node minimal implementation of a Hadoop cluster and a larger, three node system.

4.2 Iterative Development and Testing

Because iterative development is commonly used by many software architects and because of my personal experiences in using it successfully, I chose to use it for this research. Iterative development is one of many technologies which collectively make up the agile software development methodology. The list of technologies included as part of the agile process varies depending on who is asked because the process is most often defined by a set of tenants, rather than a formal definition. Of those tenants, iterative development encourages one to "deliver working software frequently" [14]. This typically means breaking the project into a set of deliverables which are each functional by themselves and can be produced in a short time period (typically one to two weeks), and in which later versions of the software add additional capability or fulfill requirements more completely. An iteration is that time period during which one of those deliverables is created. In this case, because the software architect has chosen execution time as a criterion for determining the project feasibility, they will want to measure it at the end of each iteration and thus must have an application that will execute properly in a specific environment. A number of factors were considered in determining the number of iterations required, particularly given the VMIs and Hadoop configurations discussed above.

Because we need a baseline benchmark to allow comparative performance measurements for later iterations, the first iteration was spent producing and testing the proxy legacy application described earlier. For the next iteration, because the software architect needs to be cognizant of the cost of migrating the legacy application to run in the new environment, several options would warrant consideration. One option to be considered would be to make the minimal amount of code changes required in order to be able to run the legacy application in the new environment, in this case, a Hadoop system running in a cloud. This option would give the architect a feel for the minimum amount of programming changes that would be required and because of the benchmarking in that environment, a sense for the scale of improvement (or degradation) in the execution speed. This minimal Hadoop option was implemented and tested in iteration 2.

Another option for the architect to consider in migration would be to examine making all the required application changes to take more advantage of the features of the Hadoop cloud. This option would give the architect some rough estimates of the effort required to make the legacy application capable of taking more advantage of the Hadoop system. Again, benchmarking of this iteration would provide a comparative point to both the baseline and minimal migration iterations. Recall from the earlier discussion that the architect is looking for a sizable improvement in order to be able to justify the costs of migration. Iteration 3 was spent producing and testing a regular Hadoop version of the baseline application. The applications for iterations 1, 2 and 3 were all executed in standalone mode as described earlier.

Once the iterations to complete the development and migration were finished, three versions of the application were available for further testing and benchmarking in various configurations. Since standalone operational mode was tested and benchmarked in iterations 1, 2 and 3, the architect would now want to move to the more complex configuration of pseudo-distributed

mode in iteration 4. In this iteration, only the minimal and regular Hadoop applications need to be benchmarked as the standalone Java application cannot be run in this environment. Having experimented with pseudo-distributed mode, iteration 5 was spent moving to fully distributed mode and the two node cluster configuration described previously. Again, only the Hadoop versions of the application can be run in this environment. Finally, iteration 6 examines the scalability aspects of Hadoop as discussed previously, and again, the execution environment is only appropriate for the benchmarking of the Hadoop applications. A more detailed description of the methods performed in each iteration follow.

4.2.1 Iteration 1

In this stage of testing the software architect is just starting to experiment with the cloud, Hadoop and the application. The benchmark results from the execution of the Java application in this iteration formed the baseline for all of the other benchmarks because of the similarity of the deployed configuration to many enterprise platforms. The software architect would want to compare the execution times of all the other test scenarios against this configuration in determining what, if any, benchmarking improvement was gained. These would need to be assessed against the costs of producing that stage, i.e., the cost of migration, configuration, deployment, etc.

The first iteration of the development effort produced a baseline version of the text mining application and the datasets necessary for its execution. In order to avoid issues and conflicts with intellectual property right ownership and my employer, it was deemed inappropriate to use an existing application; instead a proxy for it was sought. After some consideration, it was determined that the text-mining of server log files is a routine task of varying complexity which

can act as proxy for the processing performed by legacy applications discussed previously, but without the challenges of programming in one of languages typically associated with such applications. Like many legacy applications, a text-mining application will access one or more input files in order to perform some manipulation, processing or analysis of this input data in order to produce one or more output files.

4.2.1.1 Application development

Specifically for this thesis, such a proxy application was designed and written in the Java programming language. The application is not intended to be a generalized data-mining, but rather to accomplish a specific analysis on a specifically formatted dataset for the purpose of acting as a proxy. This application analyzes IBM WebSphere Application Server (WAS) trace/log files to determine the average time required to allocate a managed JDBC connection.

As shown in Figure 4, this application can be executed on any computer which provides a JSE JRE.

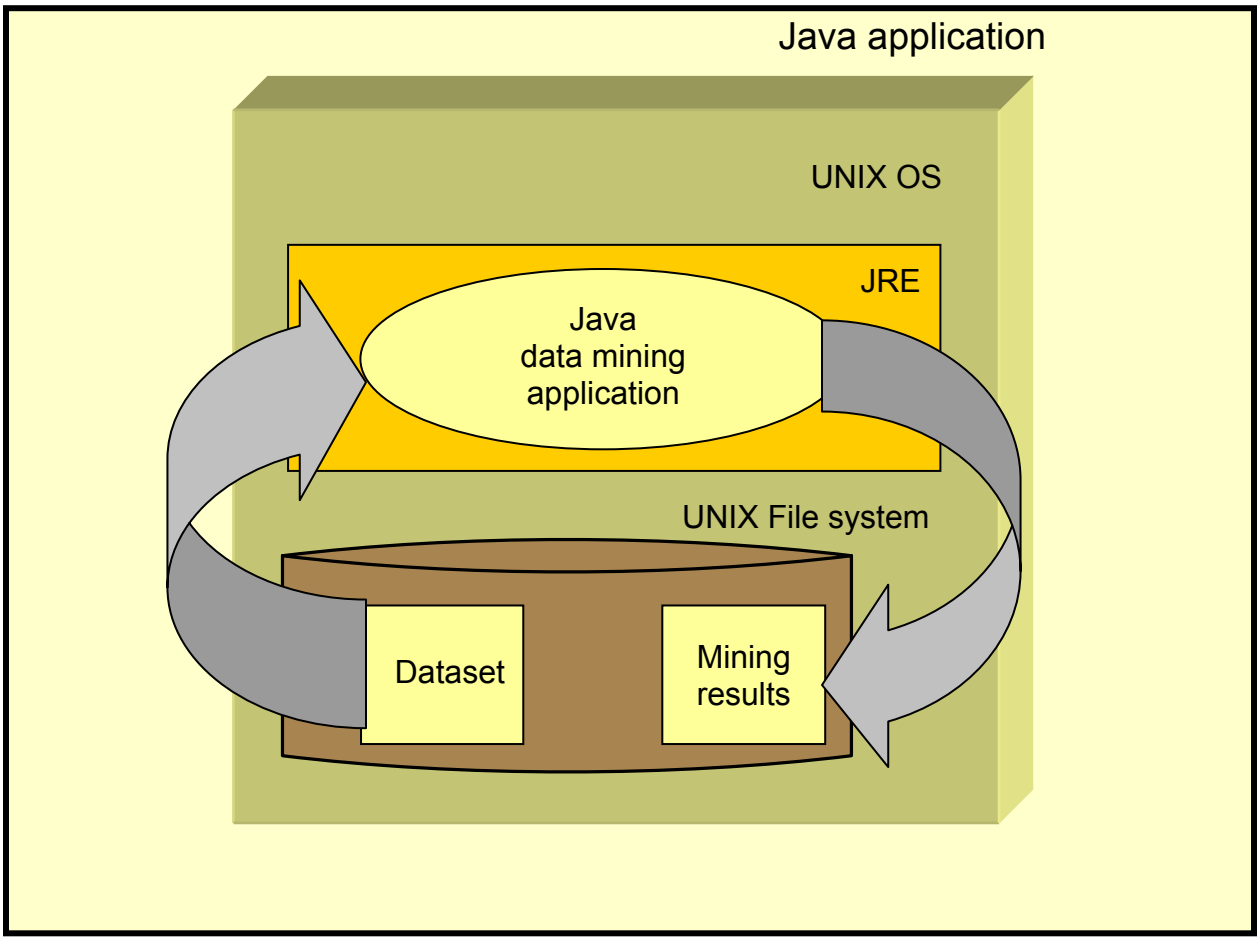

Figure 4: Baseline Java application

The proxy application reads each entry of the input trace log and calculates the average time to allocate a managed connection over the entire span of the trace file. Because the application server is multithreaded and supports multiple concurrent servlet requests, numerous connection requests and responses will appear in the log stream intermingled. In order to be able to identify the request thread associated with a particular log entry, the thread id is recorded in each entry, along with a date/timestamp with millisecond resolution. The application calculates the average time recognizing all connection request initiation and completion entries and then attempting to

match the thread ID of an initiation entry with a completion entry which follows it (proceeds temporally) in the log having the same thread ID. When it discovers such a log couplet, the proxy application then calculates the difference between the initiation and completion timestamps. The number of connection request couplets and their associated elapsed time is accumulated for the specified input trace file and when all of the records of the specified input file have been read, these statistics are written to the specified output file.

4.2.1.2 Dataset generation

Because the application performs file I/O in order to accomplish its analysis, intuitively, the size of that trace/log file (dataset) will influence the amount time spent in processing it. However, as this research measures and compares the relative execution times of applications, in order to effectively demonstrate a meaningful difference in execution time, one must either:

- demonstrate a difference of a reasonable magnitude which overshadows variable timing differences injected into the test by real-world influences like network delays influencing access times for network mounted file systems, operating system task switching, etc
- prevent the above mentioned influences from occurring or at least minimize any impacts from them
- account for the effect of the above mentioned influences

Because of the difficulty of implementing the latter two options in a parallel environment running in a cloud, the software architect may find it easiest to justify a change in the application architecture if a sizable difference in the execution speed can be demonstrated. Otherwise, if that difference can't be readily measured, it is likely that the costs either outweigh the benefits, or

that the benefits are negligible, with the end result in either case that the project will not get funded. This research attempts to address of the some of the influences on timing described above in two ways. First, in order to help average out the effect of the influences, when recording the benchmark execution times, the mean of three executions is computed. Second, this research utilizes a range of file sizes in order to examine the relationship of performance differences and dataset size. The dataset sizes chosen were:

- 20 MByte
- 2 GByte
- 5.2 GByte
- 10.5 GByte

Datasets of these sizes were obtained by collecting log files from the application server instance. To simplify the text-mining process, the application server logging and trace service was configured to produce ASCII text files. Because the analysis performed by the data-mining application involves the processing of application server trace log entries related to Java Database Connectivity (JDBC), the service was further configured by setting the trace specification level of the appropriate WAS component to "ALL". Once enabled, trace messages were generated by the application server as a result of executing a simple Java program which made a request via HTTP to a Java Enterprise Edition (JEE) application deployed on the application server. Because the JEE application uses Enterprise JavaBeans (EJBs) which are backed by container-managed persistence (CMP), a JDBC connection to a relational database is required by each servlet request. The open-source Derby database which is embedded as a component within the application server was used as the relational database. As each connection allocation is initiated and completed, a trace entry is made to the application server trace file.

For each of the dataset sizes listed above, the program making the HTTP request to the EJB simply looped until the desired log file size was obtained. The program was then stopped, the application server trace log moved to a separate location and the program restarted for the next dataset creation. Once generated, these same datasets were used in each later iteration without modification.

4.2.1.3 Test environment setup

For the first set of tests, as depicted in Figure 5, a single instance of the JRE VMI was deployed and the Java application was installed on that instance.

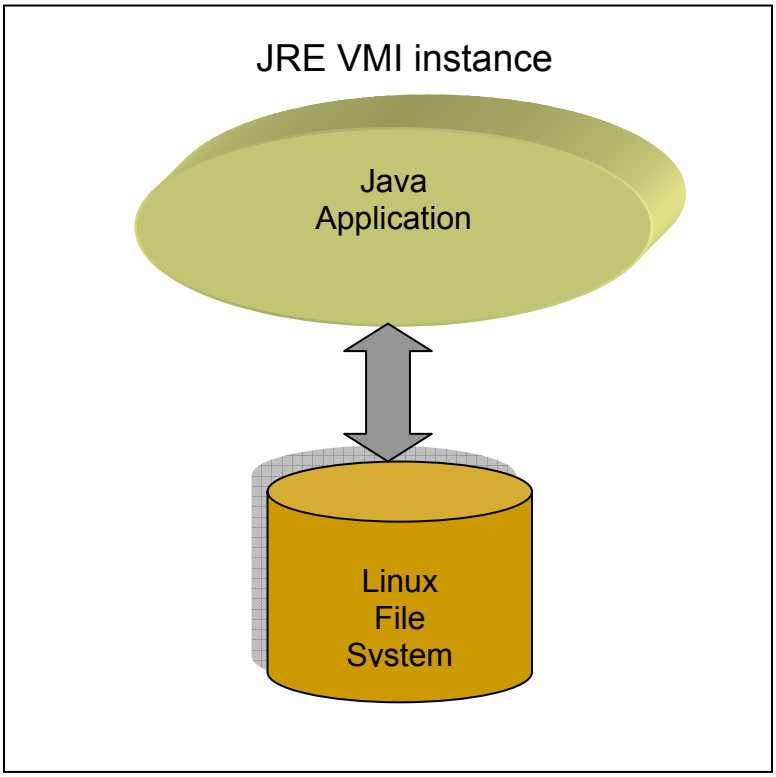

Figure 5: JRE VMI instance deployed to single node

As described previously, the JRE VMI is an appropriate environment for simulating the execution of applications targeted for the JSE JRE on a commodity-class Linux "box". The dataset files were then copied to a local file system of the VMI instance. Recall that the JRE VMI in standalone mode uses the OS file system, which in this case is the Linux file system. For each of the four dataset sizes the installed application was executed three times, for a total of 12 executions and the associated timings recorded. Then the mean of the three execution times for each dataset was computed.

4.2.2 Iteration 2

4.2.2.1 Application development

During the second iteration, that standalone application was migrated to take advantage of a utility included in the Hadoop distribution known as streaming Hadoop. This option is particularly useful for the type of rapid prototyping that the architect wants to do in the second iteration. In a nutshell, streaming Hadoop allows an application which reads input from the standard input stream of its process (stdin), performs some processing (either the Map or Reduce portion of the Hadoop MapReduce paradigm) and writes the results of that processing to the standard output stream of the process (stdout) to be deployed as a Hadoop application. Because many applications (particularly those written for the UNIX operating system) read from stdin and write to stdout, or can be converted relatively easily to do so, this utility is quite useful for quickly getting an application running within the Hadoop framework. In addition to the JSE JRE required for the first iteration, this iteration of the application requires the installation of a compatible Hadoop distribution.

4.2.2.2 Test environment setup

As depicted in Figure 6, for the second iteration tests, a single Hadoop VMI instance was deployed and both the streaming and the regular Hadoop application installed on that instance. The Hadoop configuration files were edited appropriately.

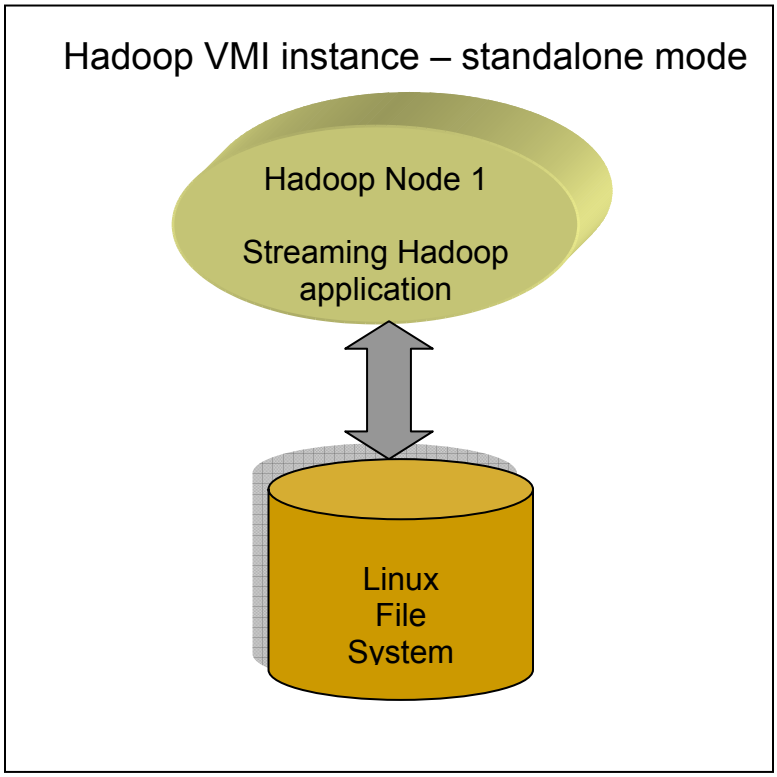

Figure 6: Standalone mode Hadoop VMI instance deployed to single node

The dataset files generated in iteration 1 were then copied to a local file system of the VMI instance. Recall that in standalone mode Hadoop uses the OS file system, which in this case is the Linux file system. The streaming Hadoop application was then executed three times for each

of the four datasets for a total of 12 executions each. The resulting execution times were recorded and the mean of the three times for each dataset computed.

4.2.3 Iteration Three

4.2.3.1 Application development

In the third iteration, the migration effort continued and the necessary changes made to the application in order for it to perform as a Hadoop application without relying on the streaming utility used in iteration 2. Even so, the minimal amount of application changes was made because the architect wants to perform just enough prototyping to be able to gather the necessary data to make a rational decision about whether the project benefits outweigh the costs. As shown in Figure 7, this iteration of the application has the same platform requirements as the previous iteration; a JSE JRE and the Hadoop distribution.

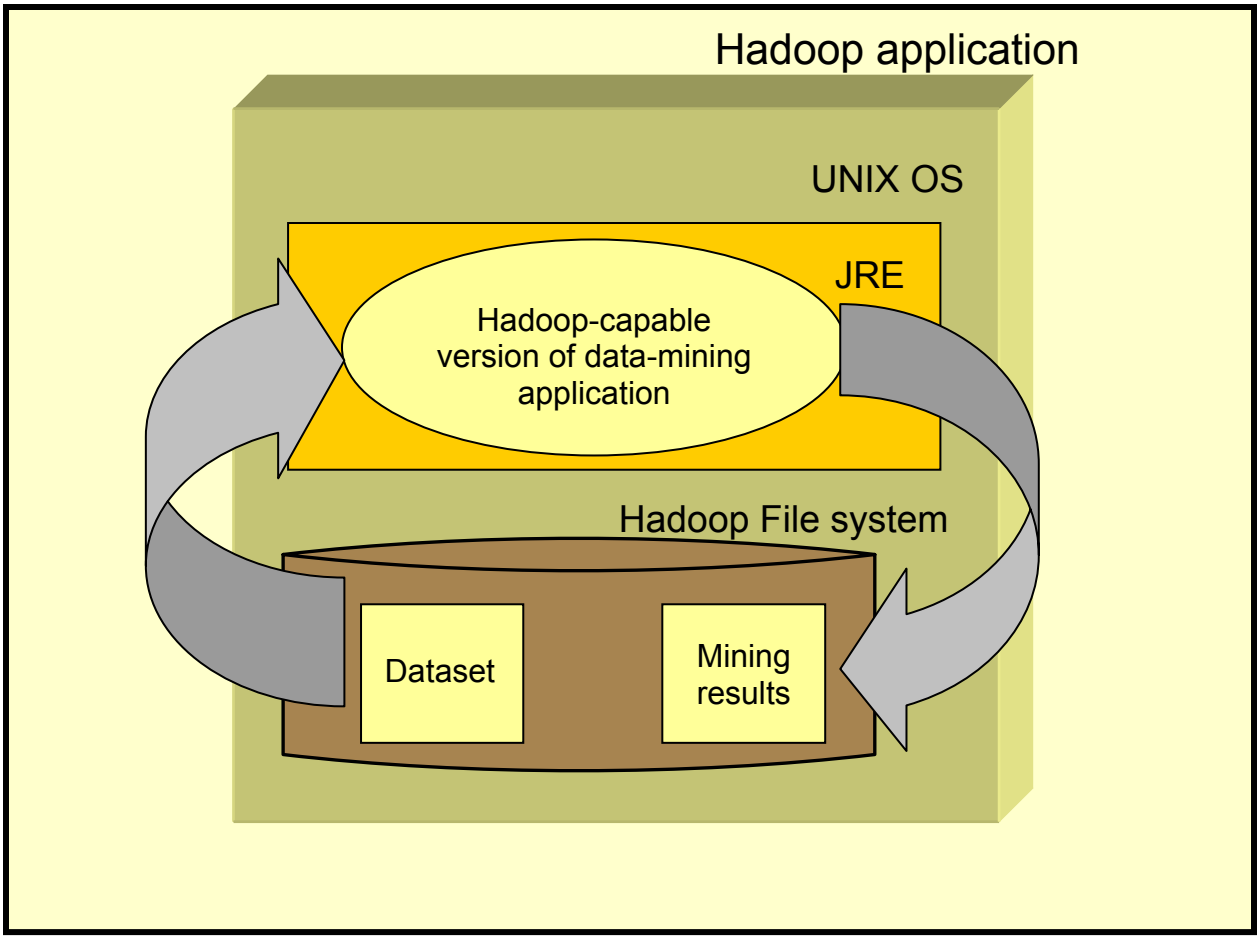

Figure 7: Migrated Hadoop application

4.2.3.2 Test environment setup

Figure 8 depicts the deployment of a single Hadoop VMI for the tests of the third iteration with the regular Hadoop application installed on that instance. The Hadoop configuration files were edited appropriately.

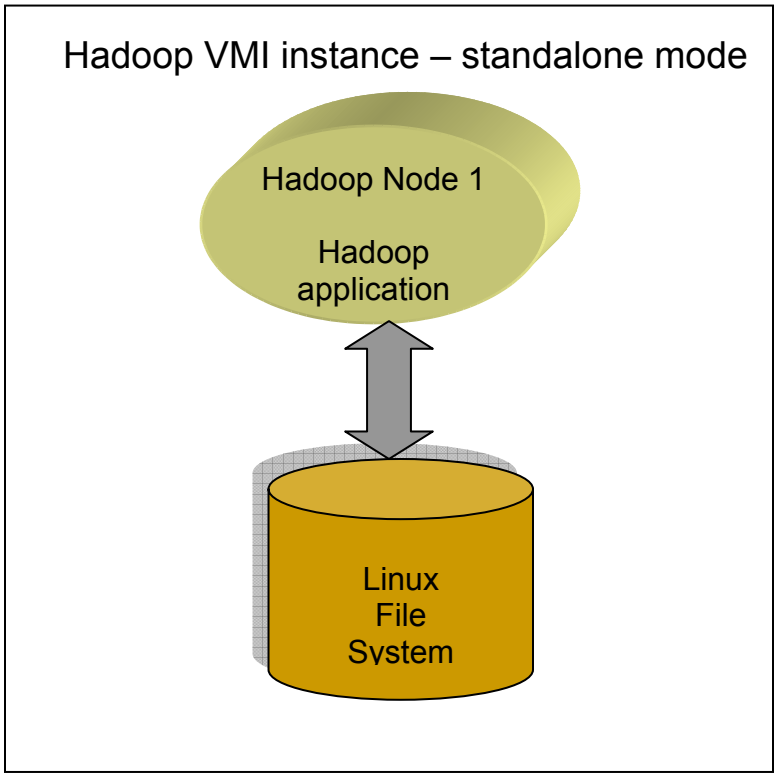

Figure 8: Standalone mode Hadoop VMI instance deployed to single node

Since in standalone mode Hadoop uses the underlying OS file system (Linux in this case), the dataset files were then copied to the VMI instance file system. The Hadoop application was then executed three times for each of the four datasets for a total of 12 executions each. The resulting execution times were recorded and the mean of the three times for each dataset size computed.

4.2.4 Iteration 4

In this stage of testing the software architect has become more comfortable with the cloud and is interested in determining if changing the configuration of Hadoop might provide some

additional benefits. One is also interested in verifying that both the streaming and regular Hadoop applications will work as expected when executed in a Hadoop cluster. Because the pseudo-distributed mode provides a cluster-like environment without the need to deploy multiple VMIs, it is well-suited for this benchmark. The software architect might choose this configuration as the first foray into Hadoop clusters because it will allow one to benchmark the application with minimal configuration. The software architect would want to compare the execution times of this test scenario to determine what, if any, benchmarking improvement was gained. These would need to be assessed against the costs of producing that stage, i.e., the cost of migration, configuration, deployment, etc.

4.2.4.1 Application development

There was no further application development completed for this iteration, instead, the streaming and regular Hadoop application versions developed in iterations 2 and 3 are employed.

4.2.4.2 Test environment setup

For the tests of iteration four, as depicted in Figure 9, a single Hadoop VMI instance was deployed and both versions of the Hadoop application were installed on the instance. The Hadoop configuration files were edited to create a two node Hadoop cluster. A two node, pseudo-distributed Hadoop system was chosen as a minimal implementation of a Hadoop cluster and was selected to provide data measuring any benefits a Hadoop application might have over a standalone Java application.

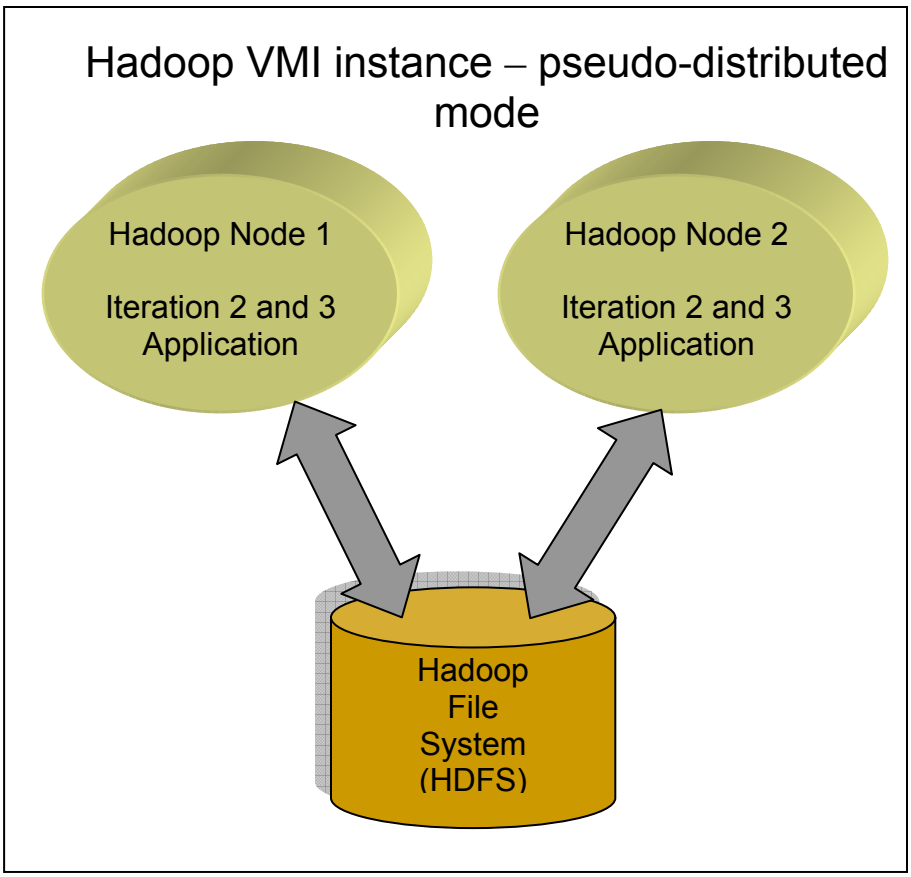

Figure 9: Pseudo-distributed Hadoop VMI instance deployed to a single node

The datasets generated in iteration 1 were then copied to the HDFS of the VMI instance. Recall from previous discussion that one of the benefits which pseudo-distributed mode has compared to standalone mode is the ability to use the HDFS. Both the streaming and regular Hadoop application was then executed 12 times, three times for each of the four datasets, and the resulting 24 execution times recorded. The mean execution time for each of the three runs per dataset was then computed.

4.2.5 Iteration 5

The potential for the full benefit of using Hadoop in a cloud was first examined in iteration 5. Again, the architect would first want to verify that both the streaming and regular Hadoop applications will execute as expected when deployed into a Hadoop cluster and then examine the execution times of several configurations. Running in fully distributed mode, each Hadoop node was deployed to its own cloud instance, the equivalent of a single processor machine. Benchmarks were measured for two different two node cluster configurations: i) an HDFS block of 64 MB, and ii) an HDFS block size of 256 MB. The first configuration was chosen to compare the performance of using HDFS in distributed mode vs. that using the Linux FS of pseudo-distributed mode with the same HDFS block size in both experiments so as to help isolate the factors causing any detected change. The second configuration was chosen in order to facilitate the comparison of using HDFS with a 64 MB blocksize against using it with a 256 MB blocksize.

4.2.5.1 Application development

Similar to iteration 4, no further application development was required for this iteration. As before, the streaming and regular Hadoop application versions developed in iterations 2 and 3 are employed for benchmarking in this iteration.

4.2.5.2 Test environment setup

For this set of tests, as depicted in Figure 10, two Hadoop VMI instances were deployed and both versions of the Hadoop application were installed on both instances.

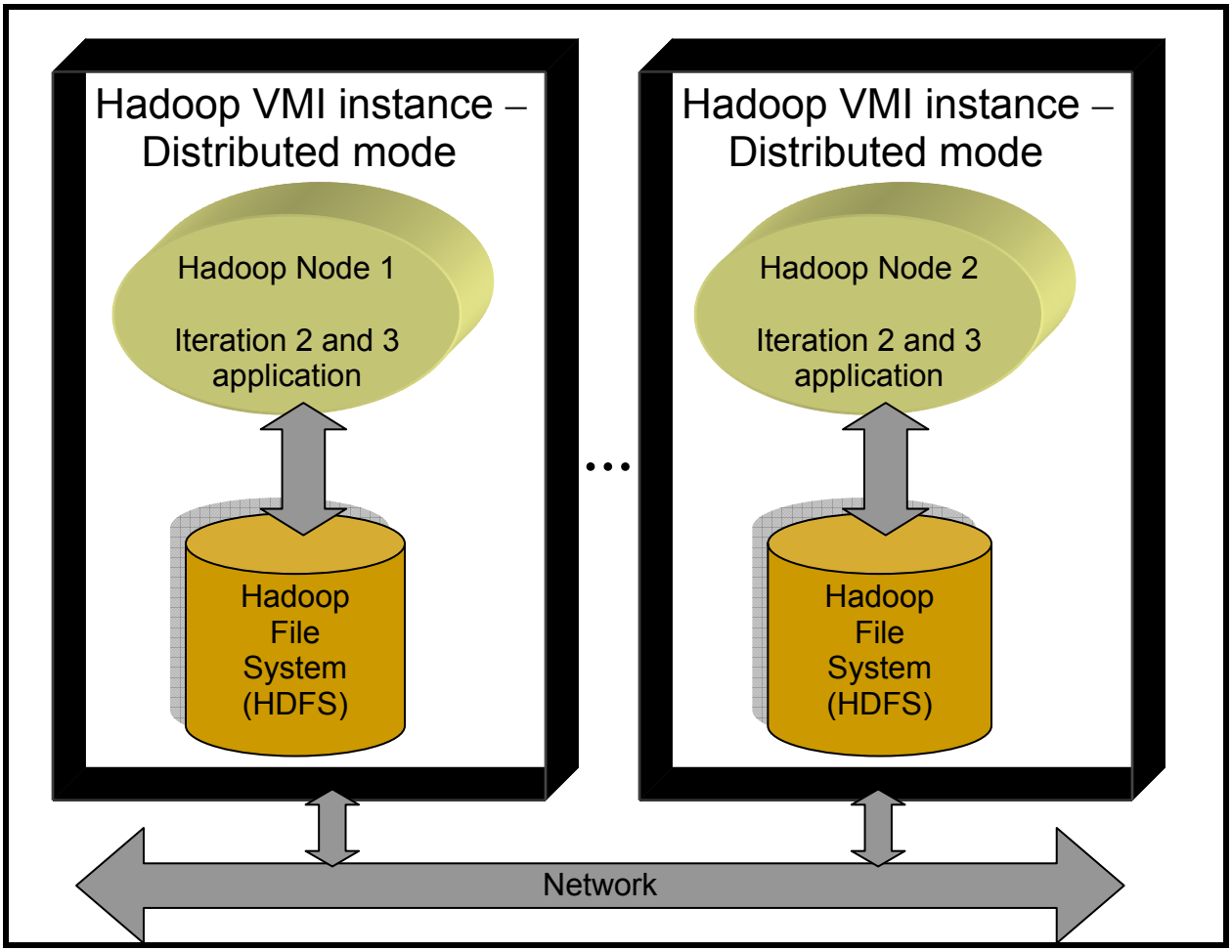

Figure 10: Distributed Hadoop VMI instance deployed to two nodes

The first group of tests in iteration 5 was run on a two node Hadoop cluster with a HDFS block size of 64 MB. The Hadoop configuration files of both nodes were edited to create this cluster. The dataset files were then copied to the HDFS. Both the streaming and regular Hadoop applications were then each executed three times using each of the four data files for a total of 24 executions. The execution times of each experiment were recorded and the mean of the three execution times per dataset computed.

Having compared differences that could be attributed to using HDFS instead of the OS FS, the next set of tests of iteration 5 examined the effect that changing the Hadoop configuration

parameters (specifically HDFS blocksize) might have on the performance. For these tests, the Hadoop configuration files were edited to change the HDFS block size to 256 MB. The dataset files were again copied to the HDFS. Then both the streaming and regular Hadoop applications were each executed three times using each of the four data files for a total of 24 executions. The execution times of each experiment were recorded and the mean of the three execution times per dataset computed.

4.2.6 Iteration 6

The potential scalability of Hadoop in a cloud was examined in iteration 6. Again, the architect would first want to verify that both the streaming and regular Hadoop applications will execute as expected when deployed into a three node Hadoop cluster and then examine the associated execution times. The three node Hadoop cluster was chosen to represent a larger version of the Hadoop cluster used in the previous iteration in order to provide data to the architect to help determine if scalability could be achieved by simply adding additional nodes to a Hadoop cluster. For the software architect, understanding the scalability aspects of the Hadoop software is critical and because of the ease with which one can deploy additional nodes into the cluster, is a worthwhile exercise.

Benchmarks were measured for a three node cluster configuration with a 256 MB HDFS block size. Running in distributed mode, each Hadoop node was deployed to its own cloud instance, the equivalent of a single processor machine. This configuration was specifically chosen to allow direct comparison of the two node cluster with 256 MB HDFS blocksize configuration of iteration 5. Because of the similarity between the two configurations, the benefits associated with adding an additional node to the cluster should be readily measurable.

4.2.6.1 Application development

The streaming and regular Hadoop application versions developed in iterations 2 and 3 are deployed without further application development.

4.2.6.2 Test environment setup

For this set of tests, as depicted in Figure 11, three Hadoop VMI instances were deployed and both versions of the Hadoop application were installed on all three instances.

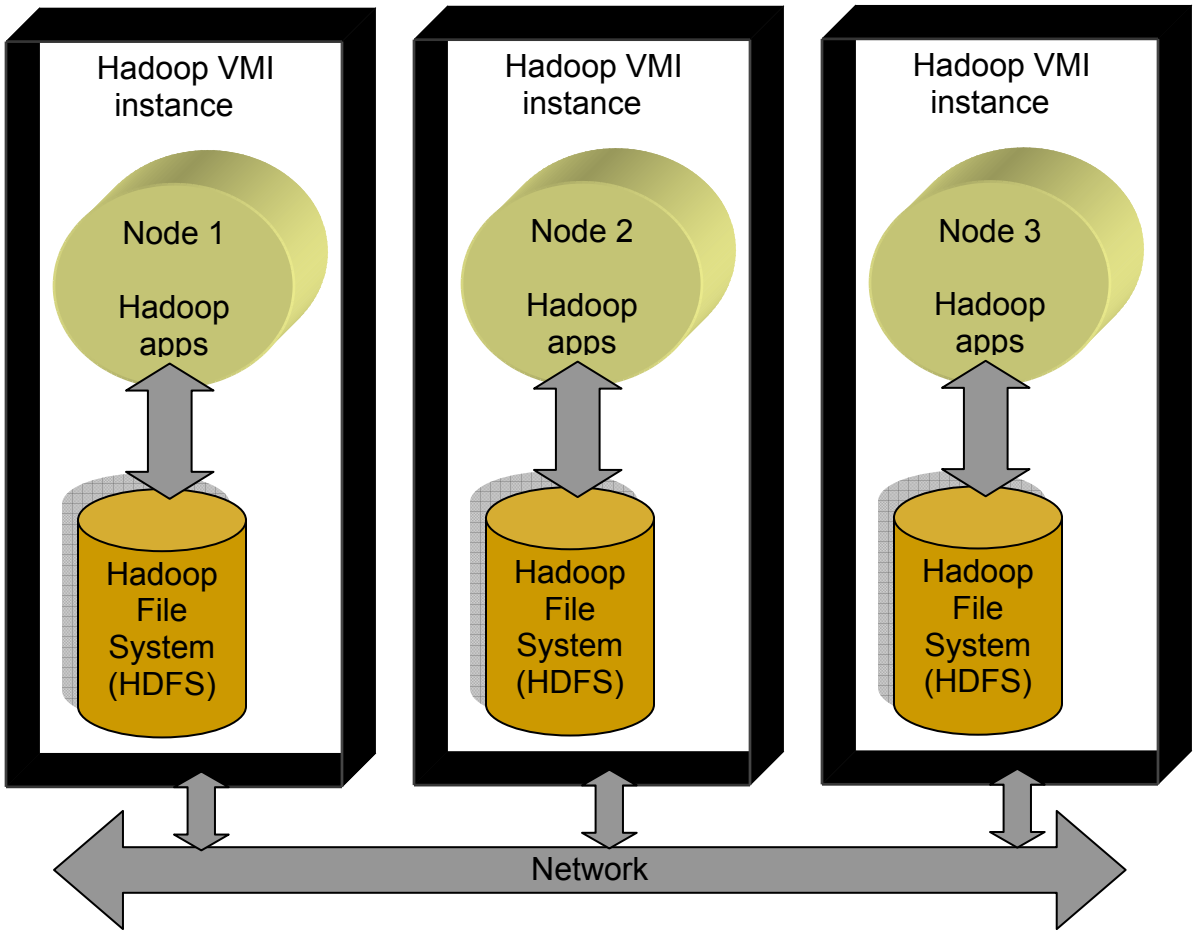

Figure 11: Distributed Hadoop VMI instance deployed to three nodes

The dataset files were then copied to the HDFS. Using each of the four datasets as input, both the streaming and regular Hadoop applications were then executed three times using for a total of 24 executions from which the times were recorded. The mean execution time for each of the datasets was computed using the three execution times measured.

4.2.7 Iteration Summary

Table 1 recaps and summarizes the configurations and applications discussed in the previous sections.

VMI type	Number of VMI instances	Number of Hadoop nodes	Hadoop operation mode	HDFS block size (MB)	Application Type	Iteration
JRE	1	N/A	N/A	N/A	Java	1
Hadoop	1	1	standalone	N/A	Streaming Hadoop	2
					Hadoop	3
Hadoop	1	2	pseudo-distributed	64	Streaming Hadoop	4
					Hadoop	4
Hadoop	2	2	distributed	64	Streaming Hadoop	5
					Hadoop	5
Hadoop	2	2	distributed	256	Streaming Hadoop	5
					Hadoop	5
Hadoop	3	3	distributed	256	Streaming Hadoop	6
					Hadoop	6

Table 1: Application benchmark and configuration matrix

Each of the cells under the "Iteration" column represents a benchmark test of the specified application running under the associated configuration. For each benchmark test, the application under test was executed a total of 12 times, three times for each one of the four dataset sizes

described previously (20 MB, 2 GB, 5.2 GB, 10.5 GB). Thus, because there are 11 benchmark test scenarios, there were a total of 132 application executions. Additionally, for each benchmark test, the mean of the three measured times was computed and recorded.

CHAPTER 5: EXPERIMENTAL RESULTS

Because the development and migration work for the text-mining application was done iteratively, the experiments for my research were carried out in a similar fashion, with increasing complexity and therefore effort for the software architect to implement. The first set of benchmarking experiments (referred to as *Standalone Mode* hereafter) was carried out on a single computing node, with either the JRE or Hadoop VMI deployed. To complete this stage of benchmarking required three versions of the application: the Java application which was written and tested during iteration 1 and both the streaming and the regular Hadoop applications, which were written/migrated and tested during iteration 2 and 3 respectively. This environment provided a well-matched test bed for the application migration which was being performed at the same time. This test-driven approach to development is another tenant of the agile process and is very effective in producing stable, releasable code (another key tenant) [14]. The concept is simple, implement a small set of features in code and test it immediately. As the functionality of the code grows iteratively, so does the test environment. At the end of this testing, the migration activity was complete and all application versions tested. These iterations measured the execution times of all three application versions in an environment which approximates a standalone alone server in a traditional enterprise computing environment. As also discussed previously, this is the only environment in which the Java application will be benchmarked since it acts as a baseline.

Once the initial development and testing was complete, the research continued in iteration 4 with another set of experiments (referred to as *Pseudo-distributed Mode* hereafter) which was performed on a single node with only the Hadoop VMI deployed. The Hadoop environment was

configured with two clustered nodes in pseudo-distributed mode meaning that all of the Hadoop daemons are started in their own JVM, but all on the same machine. Both the streaming and regular Hadoop applications were benchmarked for comparison to the previous iteration of benchmarking in order to determine what, if any benefit was gained in setting up this environment. In this iteration, the architect would be able to see how a Hadoop cluster is configured and exercise all of the web-based administrative interfaces for managing the various Hadoop processes.

The fifth and sixth iteration of experiments (referred to as *Distributed Mode* hereafter) was performed on multiple nodes and again with only the Hadoop VMI deployed. The Hadoop environment was configured with two or three clustered nodes in distributed mode meaning that all of the Hadoop daemons are started in their own machines as determined by the configuration. Similar to the previous iteration of testing, both the streaming and regular Hadoop application were benchmarked in this phase, allowing the architect to measure the performance of a Hadoop cluster and determine the ease with which scalability can be achieved. During this testing stage, several benchmarks were executed with both the default HDFS block size of 64 MB and a larger block size of 256 MB to determine the impact on the performance.

5.1.1 Standalone Mode (Iterations 1, 2 and 3)

As shown in Table 2, the raw execution times for the various dataset file sizes were recorded for each of the executions and the arithmetic mean computed for each dataset size and application.

Data set size (GB)	Test run	Execution time (seconds)			Mean execution time (seconds)		
		Java app	Hadoop app	Streaming Hadoop app	Java app	Hadoop app	Streaming Hadoop app
0.02	1	5	15	8	5	10	8
	2	5	8	8			
	3	5	8	8			
2	1	171	261	523	171	259	535
	2	171	258	557			
	3	171	258	526			
5.2	1	504	574	1377	477	583	1376
	2	464	579	1373			
	3	463	597	1378			
10.5	1	937	1158	1917	917	1099	2473
	2	920	1084	2729			
	3	892	1054	2774			

Table 2: Standalone mode benchmarks

Comparing the mean execution time to the dataset size, Figure 12 shows a fairly linear relation which one might reasonably expect.

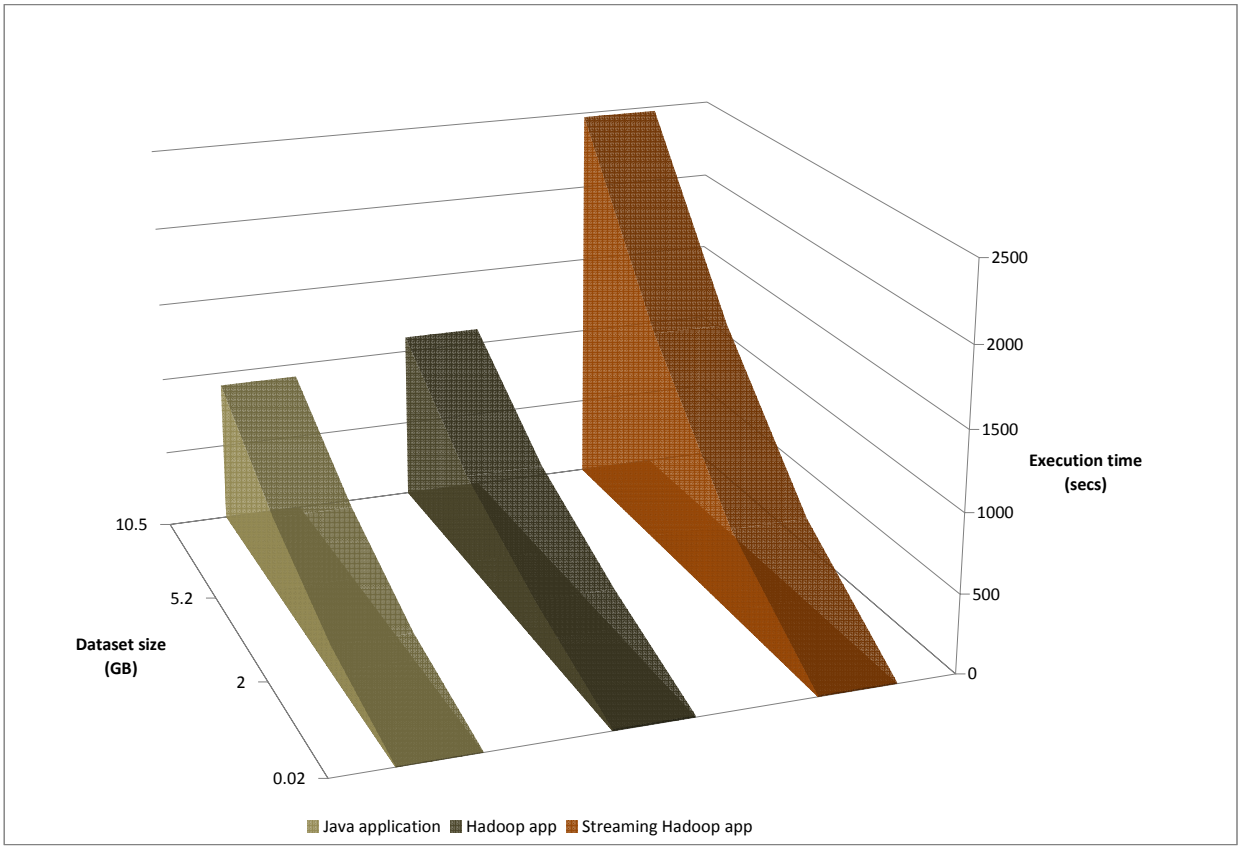

Figure 12: Standalone mode benchmark comparison

It is obvious that the standalone Java application performs better, especially on larger dataset sizes, in comparison to either Hadoop application. Table 3 shows the percentage degradation in execution time between the Java application and each of the Hadoop versions of the application in standalone mode.

Data set size (GB)	Mean execution time (seconds)			Mean degradation (%)	
	Java app standalone mode	Hadoop app standalone mode	Stream Hadoop app standalone mode	Java app vs. Hadoop app standalone mode	Java app vs. Stream Hadoop app standalone mode
0.02	5	10	8	-100	-60
2	171	259	535	-51	-206
5.2	477	583	1376	-22	-189
10.5	917	1099	2473	-20	-109

Table 3: Benchmark degradation for Hadoop apps in standalone mode

However, because the difference in mean execution time between the Java and regular Hadoop application is only about 20% for the largest dataset size, the architect could conclude that it is worthwhile to continue to the next iteration to see if the changes in configuration will result in a performance improvement. The architect would also note that while the streaming Hadoop application has the benefit of requiring little migration effort, its performance degrades to 206% in the worst case.

5.1.2 Pseudo-distributed Mode (Iteration 4)

As shown in Table 4, the raw execution times for the various dataset file sizes were recorded for each of the executions and the arithmetic mean computed for each dataset size and application. The baseline Java application data is included for comparison.

Data set size (GB)	Test run	Execution time (seconds)			Mean execution time (seconds)		
		Java app	Hadoop app	Streaming Hadoop app	Java app	Hadoop app	Streaming Hadoop app
0.02	1	5	38	44	5	38	42
	2	5	38	44			
	3	5	37	39			
2	1	171	603	860	171	610	878
	2	171	613	887			

	3	171	614	887			
5.2	1	504	1482	2198	477	1487	2201
	2	464	1480	2201			
	3	463	1500	2204			
10.5	1	937	2992	4373	917	2990	4316
	2	920	2964	4296			
	3	892	3015	4279			

Table 4: Pseudo-distributed mode benchmarks

Comparing the mean execution time to the data set size, Figure 13 shows that both of the Hadoop applications are less linear in their response than in the previous iteration.

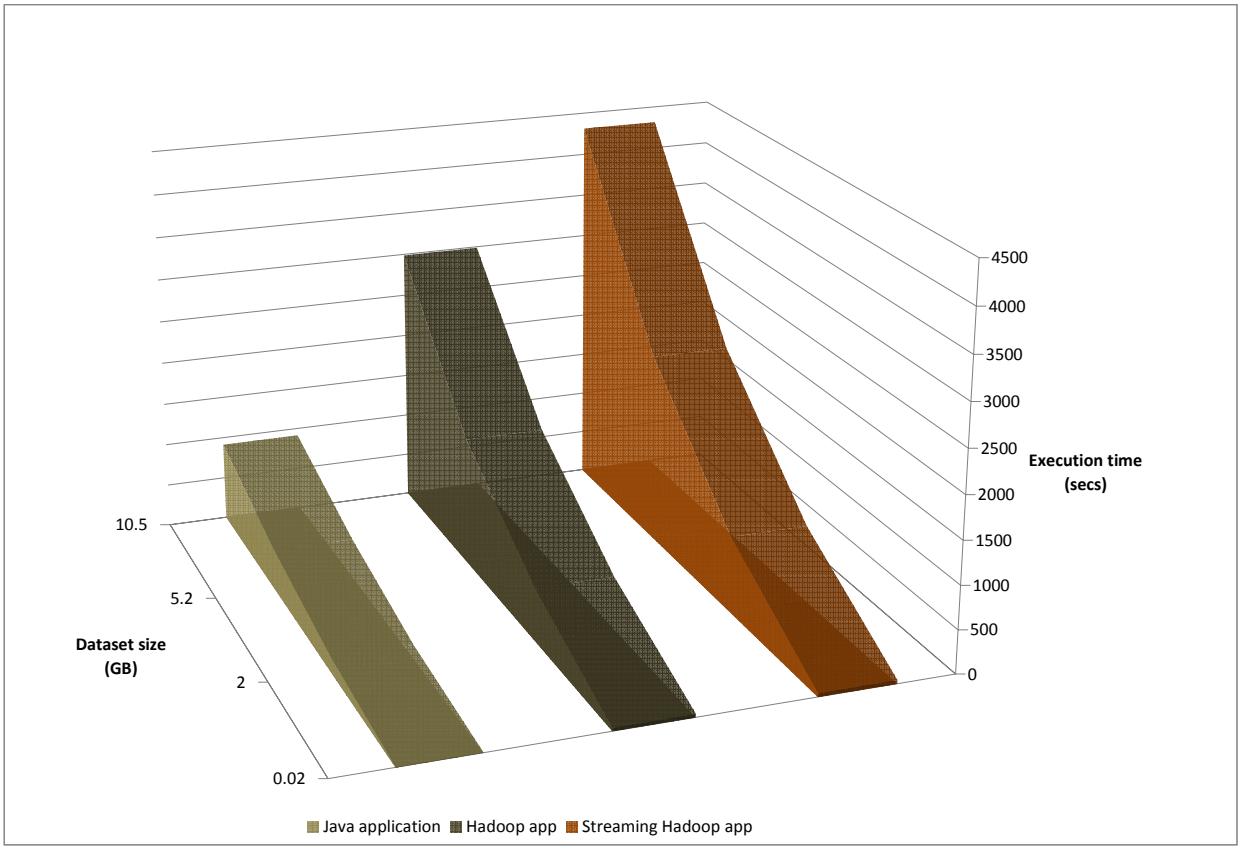

Figure 13: Pseudo-distributed mode benchmark comparison

Similar to the last iteration, the standalone Java application performs better, especially on larger dataset sizes, in comparison to either Hadoop application. However, for this iteration, the

difference in mean execution time for the regular Hadoop application has degraded significantly to become almost 3.3 times greater than that of the Java application for the largest dataset size. Worse yet, for the same dataset, the streaming version has degraded even more at a factor of 4.7. To further compare the change, Table 5 shows the percentage degradation between both versions of the Hadoop application in changing from standalone to pseudo-distributed mode. This degradation is worst for the smallest dataset and lessens somewhat as the dataset size becomes larger, but the impact is significant for all sizes and an indication that this is not a particularly useful configuration for a production environment.

Data set size (GB)	Mean execution time (seconds)				Mean degradation (%)	
	Hadoop app standalone mode	Hadoop app pseudo-distributed mode	Stream Hadoop app standalone mode	Stream Hadoop app pseudo-distributed mode	Hadoop app standalone vs. pseudo-distributed mode	Stream Hadoop app standalone vs. pseudo-distributed mode
0.02	10	38	8	42	-280	-425
2	259	610	535	878	-136	-64
5.2	583	1487	1376	2201	-155	-60
10.5	1099	2990	2473	4316	-172	-75

Table 5: Benchmark degradation for pseudo vs. standalone mode

From this, the architect would note that while the pseudo-distributed mode of Hadoop application has the benefit of easily prototyping a multi-node cluster, its performance on the largest dataset lags the Java application by a significant factor. However, the architect could also rationalize this result because of using HDFS which is intended for distributed file systems and thus apparently not as performant on a single computer and because the number of JVMs running has increased now that two Hadoop nodes and all the Hadoop daemons are all running as Java processes but on a single processor instance.

5.1.3 Distributed Mode (Iteration 5)

5.1.3.1 Two node Hadoop cluster with 64 MB HDFS blocksize

As shown in Table 6, the raw execution times for the various dataset file sizes were recorded for each of the executions and the arithmetic mean computed for each dataset size and application. The baseline Java application data is included for comparison.

Data set size (GB)	Test run	Execution time (seconds)			Mean execution time (seconds)		
		Java app	Hadoop app	Streaming Hadoop app	Java app	Hadoop app	Streaming Hadoop app
0.02	1	5	33	41	5	33	39
	2	5	33	38			
	3	5	32	38			
2	1	171	309	772	171	304	823
	2	171	300	851			
	3	171	304	847			
5.2	1	504	774	2151	477	770	2167
	2	464	769	2181			
	3	463	768	2169			
10.5	1	937	1544	4389	917	1539	4390
	2	920	1522	4419			
	3	892	1552	4362			

Table 6: Distributed mode benchmarks for 2 node cluster with blocksize = 64 MB

Figure 14 compares the mean execution time to the dataset size revealing interesting behavior for both of the Hadoop applications. Similar to the last iteration, the baseline Java application continues to perform better, especially on larger dataset sizes, in comparison to either Hadoop application. However, for this configuration the performance of the regular Hadoop application is markedly improved over that measured in pseudo-distributed mode.

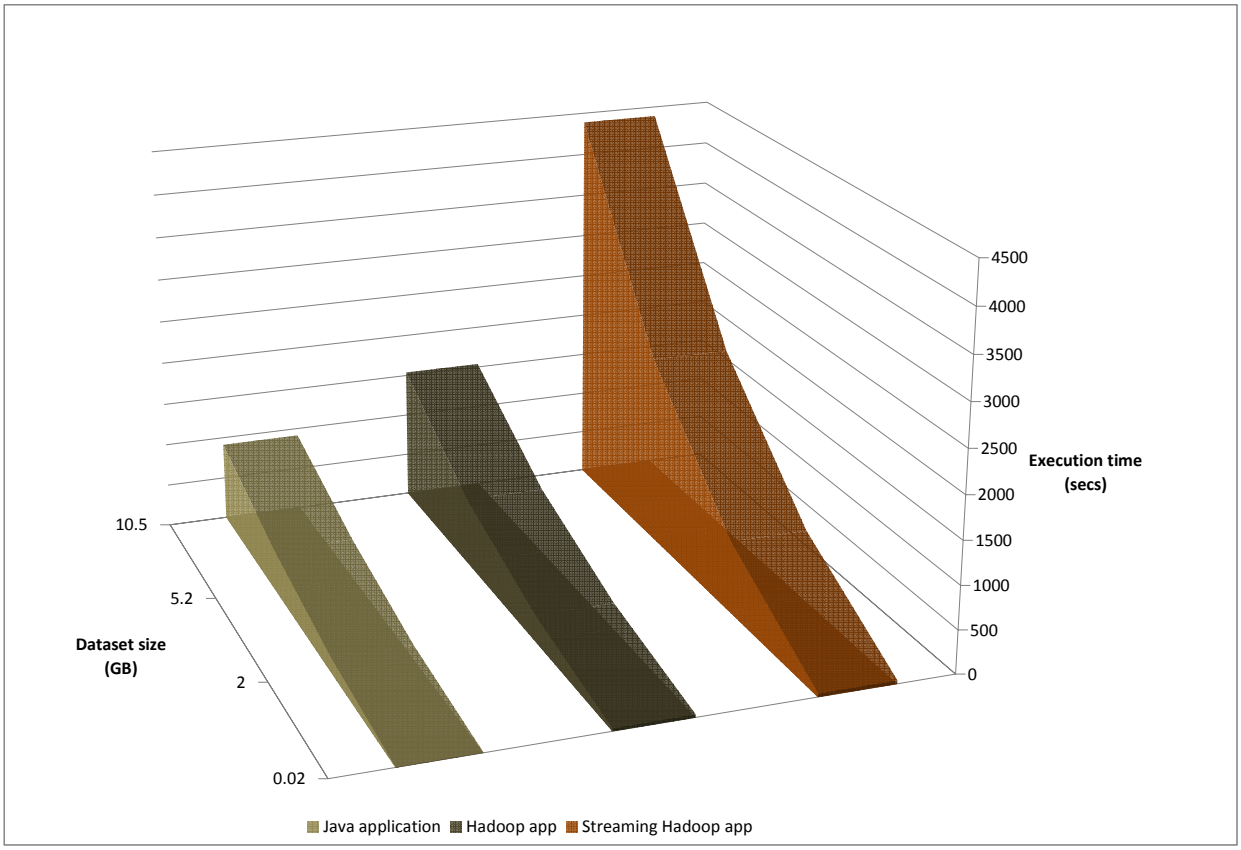

Figure 14: Benchmark comparison for 2 node cluster with blocksize = 64 MB

Table 7 reveals that the improvement ranges from 13% for the smallest dataset to 49% for the largest. In contrast, the streaming Hadoop application performance differed by a negligible amount, in fact the change appeared instead to be inversely proportional to the dataset size, ranging from 8% for the smallest to -2% for the largest, implying a slight worsening in distributed mode, but without additional larger datasets and testing, it is not possible to predict if the trend would continue.

Data set size (GB)	Mean execution time (seconds)				Mean improvement (%)	
	Hadoop app pseudo-distributed mode	Hadoop app distributed mode	Stream Hadoop app pseudo-distributed mode	Stream Hadoop app distributed mode	Hadoop app pseudo vs. distributed mode	Stream Hadoop app pseudo vs. distributed mode
0.02	38	33	42	39	13	7
2	610	304	878	823	50	6
5.2	1487	770	2201	2167	48	2
10.5	2990	1539	4316	4390	49	-2

Table 7: Mean improvement of execution time – pseudo vs. distributed mode

From this, the architect might note that using HDFS in distributed mode vs. the OS file system in pseudo-distributed mode appears to have a significant benefit for regular Hadoop applications but little impact on the streaming version.

5.1.3.2 Two node Hadoop cluster with 256 MB HDFS blocksize

Table 8 records the measured and computed arithmetic mean execution times for each of the application executions on the various dataset file sizes.

Data set size (GB)	Test run	Execution time (seconds)				Mean execution time (seconds)			
		Hadoop app 64 MB blk	Stream Hadoop app 64 MB blk	Hadoop app 256 MB blk	Stream Hadoop app 256 MB blk	Hadoop App 64 MB blk	Stream Hadoop App 64 MB blk	Hadoop app 256 MB blk	Stream Hadoop App 256 MB blk
0.02	1	33	41	59	81	33	39	90	100
	2	33	38	63	80				
	3	32	38	69	69				
2	1	309	772	201	289	304	823	309	408
	2	300	851	202	284				
	3	304	847	214	286				
5.2	1	774	2151	349	493	770	2167	504	742
	2	769	2181	325	524				

	3	768	2169	334	504				
10.5	1	1544	4389	568	858	1539	4390	847	1270
	2	1522	4419	4419	868				
	3	1552	4362	4362	866				

Table 8: Benchmark comparison for 2 node cluster 64 MB vs. 256 MB blocksize

Examining the data for both the streaming and regular Hadoop applications, several things are apparent about the 256 MB blocksize tests. First, the performance of both versions of the Hadoop application improved. Secondly, it is apparent there is a point where the performance is better for a smaller dataset size and blocksize and then similarly where performance is better for a larger dataset size and blocksize. Looking at the graph of Figure 15, that point is approximately 280 MB for the streaming application whereas for the regular Hadoop application the crossover point is approximately 2 GB. To the architect, this relationship is important to understand because it implies that a single configuration value for the HDFS blocksize may not be as performant for all dataset sizes. With this knowledge, the architect would consider the average dataset size for their environment and adjust the blocksize accordingly.

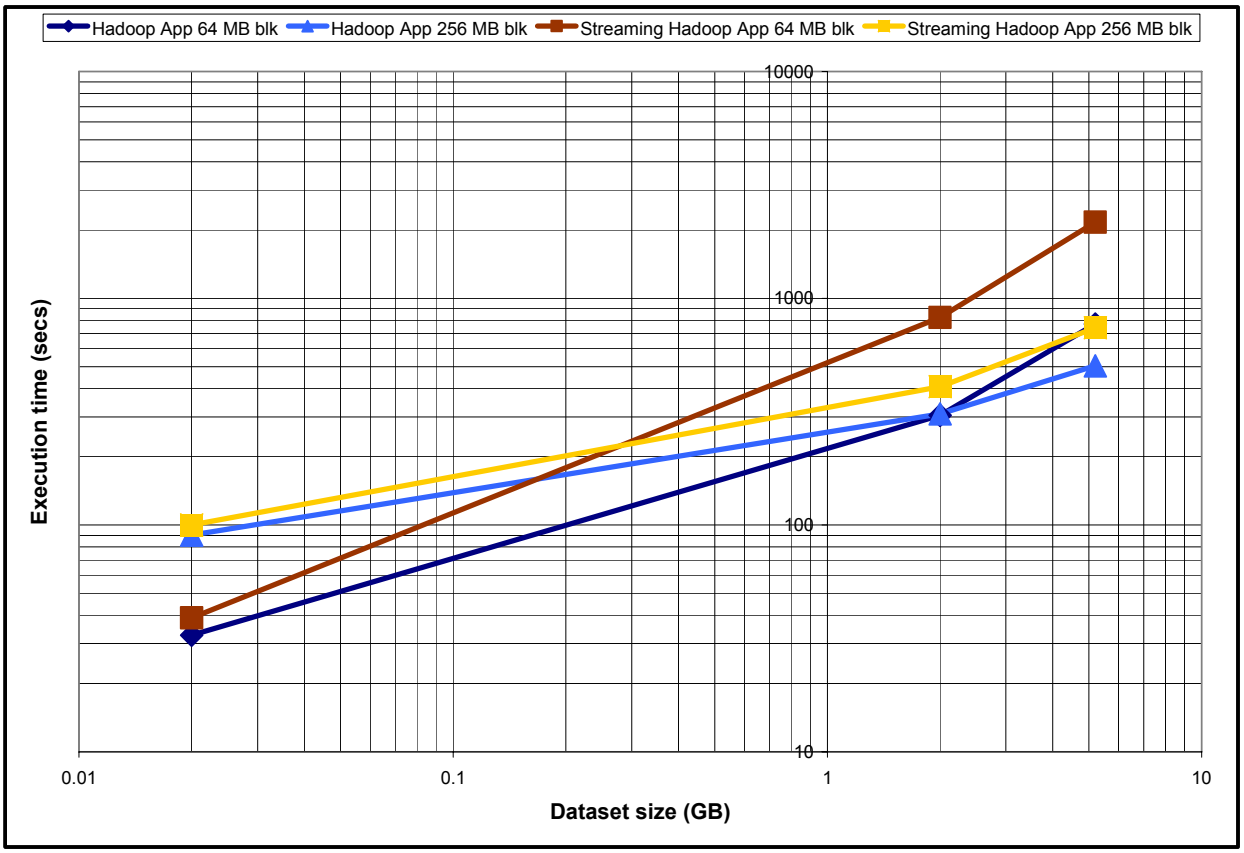

Figure 15: Crossover point for blocksize vs. dataset size

In addition to differences in benchmarks caused by the tuning of the HDFS blocksize parameter, it is worthwhile to note an additional point about one of the configurations. Table 9 shows the raw execution times and the computed arithmetic mean execution times for the various dataset file sizes for both the baseline Java application data and the two node 256 MB blocksize cluster configuration.

Data set size (GB)	Test run	Execution time (seconds)		Mean execution time (seconds)	
		Java app	Hadoop app	Java app	Hadoop app
0.02	1	5	90	5	90
	2	5	91		
	3	5	90		
2	1	171	308	171	309
	2	171	305		

	3	171	315		
5.2	1	504	514	477	504
	2	464	483		
	3	463	516		
10.5	1	937	848	917	847
	2	920	845		
	3	892	849		

Table 9: Distributed mode benchmarks for 2 node cluster with blocksize = 264 MB

Note that for the first time, the Hadoop application performance exceeded that of the standalone Java application but only after a specific point in dataset size. As seen in the graph of Figure 16, this crossover point occurs at 6.5 GB. This is significant to the architect because it the first test which has resulted in measurable performance benefits and it points out that there is a relationship between dataset size, cluster size and execution time.

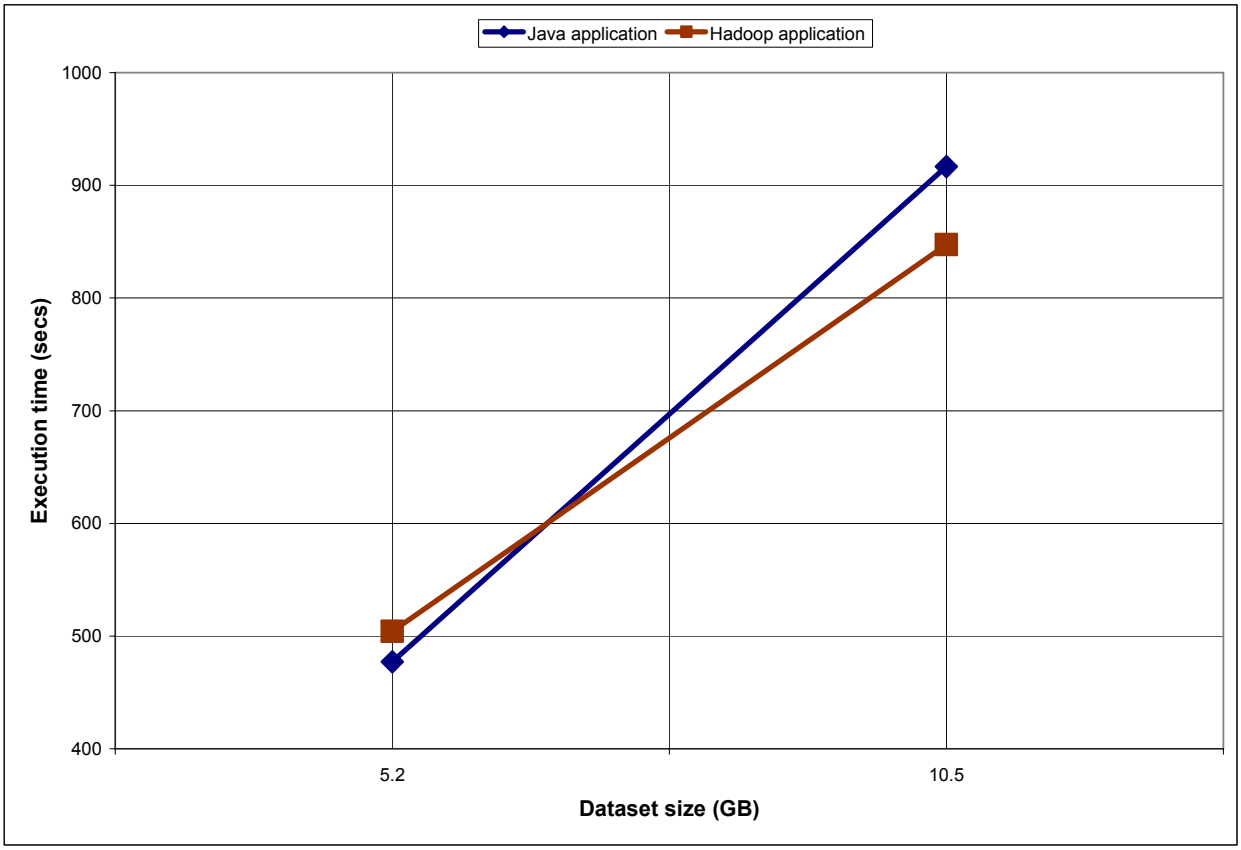

Figure 16: Crossover point for Hadoop vs. Java application

Having compared the effects of HDFS blocksize and dataset size on the application execution time in iteration 5, iteration 6 examined one of the scalability aspects of a Hadoop cloud; the effect that deploying an additional node to the cluster had on the application performance. As shown in Table 10, the raw execution times for the various dataset sizes were recorded for each of the executions and the arithmetic mean computed for each dataset size and application type. The baseline Java application data is included for comparison.

Data set size (GB)	Test run	Execution time (seconds)			Mean execution time (seconds)		
		Java app	Hadoop app	Streaming Hadoop app	Java app	Hadoop app	Streaming Hadoop app
0.02	1	5	59	81	5	64	77
	2	5	63	80			
	3	5	69	69			
2	1	171	201	289	171	206	286
	2	171	202	284			
	3	171	214	286			
5.2	1	504	349	493	477	336	507
	2	464	325	524			
	3	463	334	504			
10.5	1	937	568	858	917	569	864
	2	920	573	868			
	3	892	565	866			

Table 10: Distributed mode benchmarks for 3 node cluster with blocksize = 256 MB

Comparing the mean execution time to the dataset size, Figure 17 reveals notable behavior for both of the Hadoop applications. Similar to the benchmark for the two node cluster with a 256 MB blocksize, both of the Hadoop applications outperform the baseline Java application after the dataset size increases beyond a specific point.

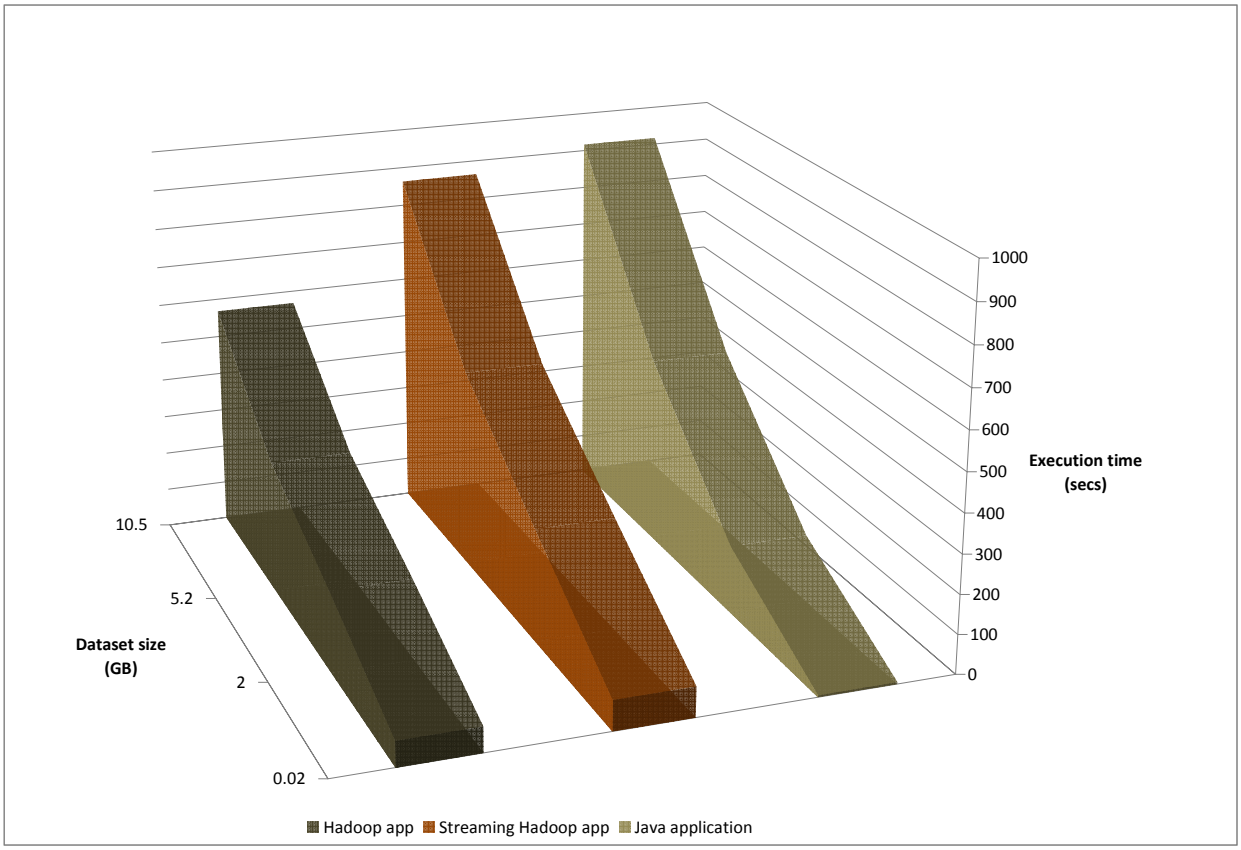

Figure 17: Benchmark comparison for 3 node cluster

Looking at Figure 18, this crossover point occurs at approximately 2.75 GB for the Hadoop application and then later at approximately 7 GB for the streaming version. To the architect, this implies that there is a minimum dataset size above which, the Hadoop application will outperform the Java application and should be considered when determining the appropriate cluster size for a given dataset size.

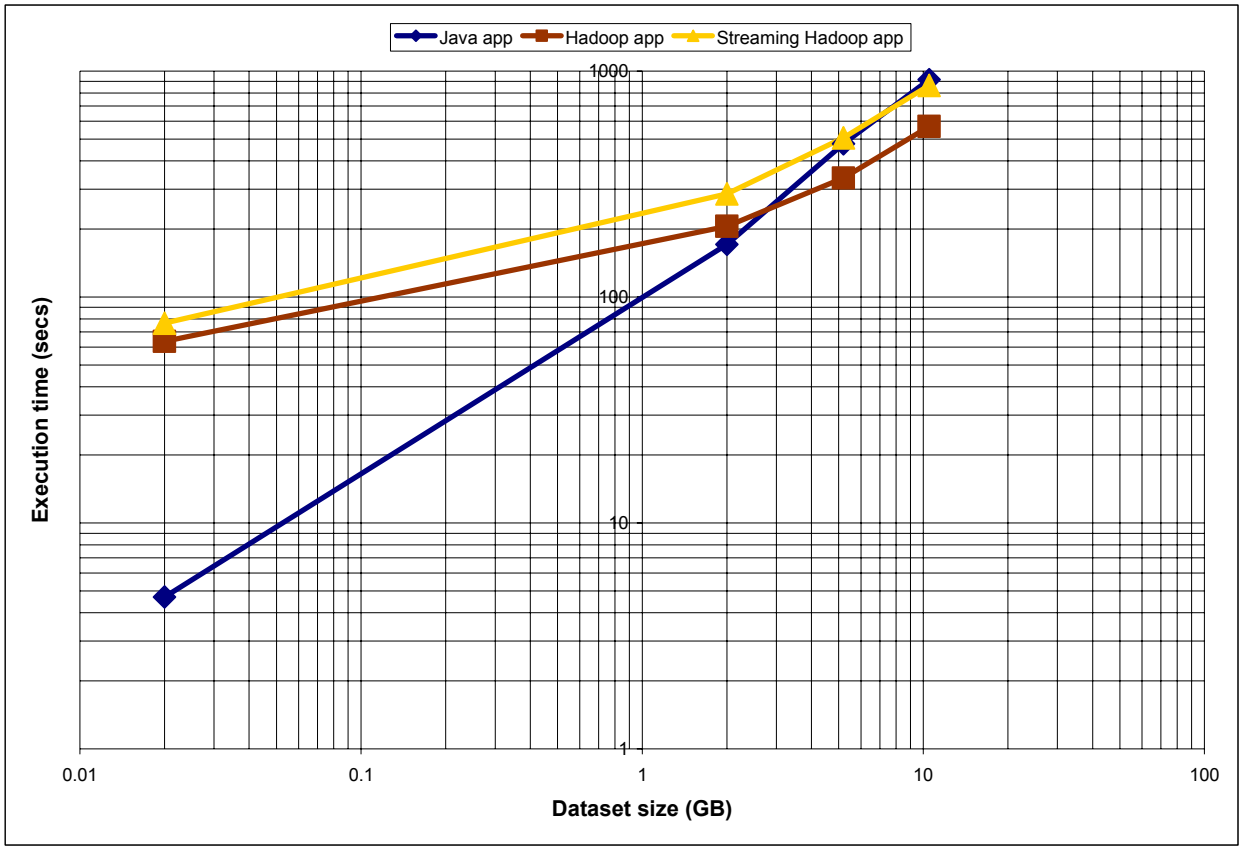

Figure 18: Crossover points for dataset size vs. execution time

Finally, having compared the three node cluster benchmarks, the architect would want to determine the amount of performance gained by adding the third Hadoop node to the cluster when compared to the two node configuration. Table 11 shows the raw execution times and the computed arithmetic mean execution times for the various dataset file sizes for both Hadoop applications executing in the two different cluster sizes.

Data set size (GB)	Test run	Execution time (seconds)				Mean execution time (seconds)			
		Hadoop app 2 node cluster	Hadoop app 3 node cluster	Stream Hadoop app 2 node cluster	Stream Hadoop app 3 node cluster	Hadoop app 2 node cluster	Hadoop app 3 node cluster	Stream Hadoop app 2 node cluster	Stream Hadoop app 3 node cluster
0.02	1	90	59	101	81	90	64	100	77

	2	91	63	100	80				
	3	90	69	99	69				
2	1	308	201	413	289	309	206	408	286
	2	305	202	412	284				
	3	315	214	398	286				
5.2	1	514	349	735	493	504	336	742	507
	2	483	325	739	524				
	3	516	334	751	504				
10.5	1	848	568	1271	858	847	569	1270	864
	2	845	573	1265	868				
	3	849	565	1273	866				

Table 11: Benchmark comparison of 2 vs. 3 node cluster

From this table, it is clear that the performance of the three node cluster is better than that of the two node cluster for all dataset sizes and for a given application type. Unlike some of the other comparisons, this means the comparative benefit of adding an additional node to a cluster does not appear to be dependent on dataset size and is thus more straight-forward to quantify. Table 12 shows that for the regular Hadoop application, adding an additional node provided a range of 29-33% performance gain, while for the streaming version, the gain ranged from 23-32%. It may be worth noting that the gains were slightly smaller for the smallest dataset size.

Data set size (GB)	**Mean execution time (seconds)**				**Mean improvement (%)**	
	Hadoop app 2 node cluster	**Hadoop app 3 node cluster**	**Stream Hadoop app 2 node cluster**	**Stream Hadoop app 3 node cluster**	**Hadoop app 2 node vs. 3 node**	**Stream Hadoop app 2 node vs. 3 node**
0.02	90	64	100	77	29	23
2	309	206	408	286	33	30
5.2	504	336	742	507	33	32
10.5	847	569	1270	864	33	32

Table 12: Mean improvement of execution time – 2 vs. 3 node cluster

CHAPTER 6: DISCUSSION OF RESULTS

As stated, my research progressed iteratively through a logical sequence of steps which the software architecture could use in planning a project migrating traditional enterprise applications to utilize parallel computing in a cloud environment. My work started with the question of whether there were benefits that could be gained in making such a migration and if so, what were some of the costs of doing so and could they be offset by the benefits. To address those questions, the first step is to review whether there were any measured benefits. I proposed that using the benchmarked execution times for an application, acting as proxy for a legacy application that can be found in any corporate IT system, could be an indicator of whether or not the amount of computing resources to complete a routine task could be reduced. If one accepts that a reduction of computing resources logically would result in lower expenses in performing that routine task, then it follows that a shorter execution time for the application will save money over a period of time.

Looking back at the results, iterations 1-3 showed that executing either the streaming or regular versions of the Hadoop application offered no improvement on execution times over the baseline Java application in standalone mode, so it is not likely that either version of the application could justify the cost of the migration in that mode of operation. In the best case scenario for standalone mode, the regular Hadoop application on the largest dataset took 20% longer to execute. While that doesn't represent a huge disparity, overall, that set of experiments didn't indicate that stopping after iteration 3 would provide measurable benefits based solely on execution times. There were some benefits that could be gained by performing those tests though. As a part of an iterative development plan, the proper operation of both versions of the

Hadoop application in a cloud environment was verified, implying that either could be delivered and would perform the expected data-mining, but not in as timely manner as the existing application. Thus, these benefits would probably only be recognized as an initial phase of the overall planned migration effort. Another benefit could be the potential for cost savings based on using the cloud computing resources on demand. As discussed earlier, this could alleviate the cost overhead of maintaining the systems while not actively performing data-mining, but depends on several factors. If the corporate IT organization provides cloud resources at little or no cost to the department of the architect, then it is likely one could save money. Otherwise, the calculation of the true cost of obtaining the cloud resources had to be considered. Since in standalone mode all three applications benefited from running in a private cloud, one could logically argue that of the three application versions, if one could procure cloud resources at little expense, the standalone Java application would be the most economical scenario of the three. In that case, the architect might be able to justify migration of the environment in which the application is executed but not migrate the application to Hadoop.

Examining the results from the experiments of iteration 4, as in the earlier iterations, neither version of the Hadoop application outperformed the baseline benchmark of the Java application. Because of the large degradation in execution times of both Hadoop applications, there are no benefits to be gained with respect to execution time. However, similar to the previous iterations, there are lessons to be learned from the iteration. Verifying that the migrated applications will function properly in a Hadoop cluster environment is necessary and was readily accomplished by the fourth iteration. Additionally, the architect gained a better understanding of the configuration changes needed to set up the cluster and that experience is needed for the next iteration. Given that no development effort was expended during iteration 4, the cost of completing the iteration

was largely the time to deploy the proper VMIs and some minimal editing of the Hadoop configuration files. That seems a low cost for the benefit of quickly verifying the proper operation of the applications in a Hadoop cluster. The other lesson learned from iteration 4 is that one cannot judge the eventual performance of an application by running it in pseudo-distributed mode. Based on these factors, like the earlier iterations, expending the resources to complete iteration 4 is only justifiable as a phase in a larger plan, but that is often the case for projects developed iteratively.

Finishing the discussion with the results for iterations 5 and 6, the architect finally has the performance numbers to quantify a benefit from the migration project. With a dataset larger than 6.5 GB, even a two node Hadoop cluster running the non-streaming version of the Hadoop application can match the baseline benchmark of the Java application and surpass it with increasing dataset size. However, looking back at the results from that benchmark, the difference in performance at the largest dataset size is still less than 10%. Recall from our earlier discussion that a sizable difference in performance was the goal, as it would make the cost justification easier, but <10% probably doesn't qualify as a sizable difference. However, if that cost reduction could be combined by savings based on using a cloud instead of dedicated resources, the justification may be easier. At the same time, based on the results of iteration 6, the architect would find the justification even easier because the performance differences grow sizably for the largest dataset. Comparing a two node and a three node cluster, the performance of both Hadoop applications was improved in the range of 30% by simply deploying and configuring an additional node to the cluster. With a dataset larger than 2.75 GB, the non-streaming version of the Hadoop application executing on a three node Hadoop cluster can match the baseline benchmark of the Java application and surpass it with increasing dataset size.

Assuming that the cost of deploying another Hadoop VMI instance for the additional node was incremental, the cost might be offset by the overall 38% improvement that could be achieved when comparing the baseline benchmark of the Java application and the Hadoop application for the largest dataset. That 38% increase is a quantifiable benefit which translates into the sizable difference which the software architect was seeking in order to make the justification more straight-forward. Since no additional migration work was required for either iteration 5 or 6, there is no development cost associated with them. As mentioned earlier, the only additional cost is the incremental cost of deploying and configuring an additional Hadoop VMI instance.

It is worthwhile to note, although in comparison the streaming Hadoop version was only approximately 6% faster than the baseline benchmark of the Java application for the largest dataset, that version does have a tangible advantage over the regular Hadoop application. The cost of migrating the Java application to the streaming Hadoop version, as discussed earlier, was minimal, and if the datasets with which the architect need to work are large enough (>7 GB), it is possible that they could justify the cost of the proposed project by doing just the migration work of iteration 2 and deploying the streaming Hadoop application into a 3 node (or larger) Hadoop cluster.

Another less tangible benefit of iteration 5 and 6 is the experience with Hadoop clustering gained by the architect and the recognition, based on the experiments, that some configuration parameters like the HDFS blocksize may have a dramatic effect on the performance of the application with regards to dataset size and may require additional benchmarking to optimize the setting for a given dataset size. Similarly, for the applications tested, there are a minimum number of nodes required for the Hadoop cluster performance to exceed that of a standalone application, this number is best determined by experimentation. Given the relative ease of

deploying additional nodes to a Hadoop cluster and the benefit gained from doing so, it seems that adding additional nodes to the cluster is worthwhile. However, it is possible that there is a point of diminishing returns, where the cost of adding another node (which includes the cost of the VMI instance lease, if any, and the cost of configuration) is outweighed by the improvement, but additional benchmarking would be required to make that determination.

CHAPTER 7: REFERENCES

[1] J. Foley, "10 Cloud Computing Predictions", InformationWeek, no. 1219, Feb 02, p. 20.

[2] M.R. McBride, "The Software Architect", Communications of the ACM, vol. 50, no. 5, pp. 75-81.

[3] P. Louridas, "Up in the Air: Moving Your Applications to the Cloud", IEEE Software, vol. 27, no. 4, pp. 6-11.

[4] H. Frank Cervone, "An overview of virtual and cloud computing", OCLC Systems & Services, vol. 26, no. 3, pp. 162-165.

[5] H. Jin, S. Ibrahim, T. Bell, L. Qi, H. Cao, S. Wu and X. Shi, "Tools and Technologies for Building Clouds", Computer Communications and Networks, 2010, vol. 0, part 1, pp. 3-20.

[6] P. Mell and T. Grance, "The NIST Definition of Cloud Computing", Communications of the ACM, vol. 53, no. 6, p. 50.

[7] R. Prodan and S. Ostermann, "A survey and taxonomy of infrastructure as a service and web hosting cloud providers", 2009 10th IEEE/ACM International Conference on Grid Computing, pp. 17-25.

[8] Amazon Corporation, "Amazon Elastic Compute Cloud (Amazon EC2)", vol. 2011, no. 09/05.

[9] IBM Corporation, "IBM Cloud Computing: Overview", vol. 2011, no. 09/06.

[10] S.E. Arnold, "MapReduce, Chubby and Hadoop", KM World, vol. 19, no. 10, pp. 1-18.

[11] M. Bhandarkar, "MapReduce programming with Apache Hadoop", 2010 IEEE International Symposium on Parallel & Distributed Processing, p. 1.

[12] K. Shvachko, Hairong Kuang, S. Radia and R. Chansler, "The Hadoop Distributed File System", 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies, pp. 1-10.

[13] V. Tran, J. Keung, A. Liu and A. Fekete, "Application Migration to Cloud: A Taxonomy of Critical Factors", 33rd International Conference on Software Engineering, pp. 22-28.

[14] M. Fowler, "The Agile Manifesto", Software Development, vol. 9, no. 8, 08/01/2001, pp. 28- 32.

[15] T. White, "Hadoop: The Definitive Guide", O'Reilly Media, 2nd edition, 2009.