LARGE SCALE DISTRIBUTED COMPUTING IN THE CLOUD

Agron Cela

Bachelor of Engineering
Software Engineering

Department of Software Engineering
Macquarie University

September 5, 2016

Supervisor: Prof. Young Choon Lee

ACKNOWLEDGMENTS

I would like to acknowledge my family for the support and encouragement that they have provided throughout my research thesis.

Special thanks to Prof. Young Choon Lee for facilitating this project and providing guidance throughout the semester.

STATEMENT OF CANDIDATE

I, Agron Cela, declare that this report, submitted as part of the requirement for the award of Bachelor of Engineering in the Department of Software Engineering, Macquarie University, is entirely my own work unless otherwise referenced or acknowledged. This document has not been submitted for qualification or assessment an any academic institution.

Student's Name: Agron Cela

Student's Signature: Agron Cela (electronic)

Date: November 7, 2016

ABSTRACT

Cloud computing is a rapidly growing paradigm that facilitates the ability for scalable and distributed graph drawing algorithms. Platforms such as Amazon Web Services offer virtually unlimited cloud resources in a pay-as-you-go fashion. Evidently, on demand availability of abundant resources accessible enables large-scale processing in a cost effective approach. There are numerous open source systems that provide distributed graph processing platforms. Frameworks such as Apache Giraph and Spark provide efficient tools for large-scale algorithms.

Boundless cloud facilities create opportunities for optimization methodologies for improving the efficient use of cloud services while concurrently reducing the overall cost. Research projects and business applications have turned towards cloud distributed processing. The ability to remain within budget constraints while maximizing execution speeds is of the essence.

Optimizing resource scheduling and provisioning are the underlying fundamental ways of efficiently maximizing processing times while reducing cost. Distributed algorithms such as graph drawing require large computational facilities to process graphs with millions of edges. In this research project, we aim to examine and introduce models to efficiently exploit cloud service abundance using graph drawing algorithms for scheduling and resource provisioning problems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The rise in cloud computing platforms such as Amazon Web Services, Microsoft Azure and Google Cloud have evolved the way data is processed. The readily available abundance of cloud resources has drastically transformed the potential for developers to innovate their ideas as it's no longer required to have large capital outlays in hardware for computationally intensive processes [5].
Platforms such as Amazon EC2, offer pay-as-you-go resources to facilitate users with easily accessible computing power without the need of substantial initial funds. With the flexibility of cloud computing, resources can be hired on a demand basis providing scalable solutions.
Traditionally, big data processing required the use of powerful processing units that are costly and time intensive. However, through the use of distributed frameworks it is possible to utilize large scale distributed processing through the use of machine-clusters. Frameworks such as Apache Giraph, GraphX and GraphLab provide powerful distributed algorithms for processing large graph data. With rise in available resources it is possible to process large graph data in a timely manner. Despite this availability, concerns with resource over-allocation as well as the cost of hiring machines can create gaps where efficient resource scheduling techniques can drastically reduce the overall project cost and resource usage. Thus, optimization factors are a great concern amongst large graph drawing (processing) applications.

Algorithms have been developed to take advantage of distributed hire-as-you-go resources. However, as the amount of data is ever increasing, new methodologies and algorithmic solutions have been developed. Such algorithms include graph drawing/visualization as well algorithms based on social and web graph processing. Despite the efficiency of graph processing algorithms, resource allocation techniques have been developed to support several work-sharing techniques through graph partitioning. Further, frameworks such as Apache Giraph take advantage of resource abundance and efficient resource allocation in order to process graphs with millions of edges. Graph processing entails the computation of thousands of tasks through the use of concurrent computations. Factors such as network communication, resource competition and transmissions between parallel processes can

affect the overall performance of the mentioned algorithms.

1.1 Project Goal

The goal of this project is to develop efficient resource allocation and scheduling solutions for graph processing algorithms on the Apache Giraph framework. Specifically we are aiming to identify methods of optimizing the execution speeds, resource consumption, and overall costs associated with running these algorithms.

To accomplish this, factors that affect the performance of graph processing algorithms must be taken into consideration. Once an adequate understanding of such factors has been procured, we can begin to form techniques that address these factors and improve the overall efficiency of execution. Graph processing algorithms including graph drawing and visualization will be conducted as a case study in these experiments. With the conclusion of this thesis, improvements on the overall performance of such algorithms will be achieved.

Chapter 2

Background and Related Work

2.1 Distributed Cloud Processing Models

With the rapid increase in the number of devices globally and with people being connected digitally more than ever before, the amount of data being produced and the need to store/process it is ever increasing. With these challenges, graphs have become a great model for displaying and absorbing information from large data-sets.

Initial frameworks such as MapReduce [3] provide a simple programming model for building scalable parallel algorithms to process large amounts of data on clusters. However, new frameworks have improved the need to design iterative jobs manually by chaining multiple MapReduce tasks, as well as Hadoop extensions such as HaLoop [30], Twister were introduced to optimize the iterative design of MapReduce.

Graph processing platforms based on the Pregel type frameworks such as Pregel, Giraph, Mizan, GPS, Pregilx based on the (BSP) Bulk Synchronous Parallel system [27] [29] which uses a vertex centric programming model with computations on vertices being represented as sequences of supersteps.
Each superstep consists of synchronizations between the nodes participating at superstep barriers while each vertex has an active or inactive state for each iterations/superstep [15]. Pregel was introduced by Google the first BSP implementation specifically for programming graph algorithms using "think like a vertex" paradigm [15].

Numerous systems have been developed to exploit the optimization benefits of Pregel. Apache Giraph was implemented in Java to work on top of the infrastructure of Hadoop framework. Giraph is designed to run processing jobs as map-only jobs on Hadoop and utilizes the Hadoop HDFS for data input and output. Other BSP based systems such as Apache Hana focuses not only on graph processing, but also other applications such as algebra and matrix computations.

Other systems such as GraphLab [9] implemented using C++ which differently from

Pregel and Giraph and GraphLab depends on shared memory abstraction and the (Gather, Apply, Scatter) processing model [22]. Alternative graph processing platforms such as GraphX, GraphChi, PowerGraph all are suitable platforms for conducting further research into graph drawing. Thus, the benefits of each platform have to be taken into consideration. According to experiments performed [22] it is difficult to determine a single best platform for graph visualizations. Thus, during

2.2 Introduction to AWS EC2 Performance

Amazon EC2 provides several options of virtual machines that are provided with customisable specifications. These virtual machines' specifications are called instances. Every instance comes with its specific configurations, this includes the memory amount, CPU power and storage. The importance of these configurations is the fact that specific configurations can be set in order to optimize the cost effectiveness and efficiency.

From a first glance, it is clear that when using instances form the Amazon EC2 platform, the more you pay results in better instance configurations. In turn, applications should theoretically increase in performance proportionally. However, this is not necessarily true as adding more instances or using machines with more available CPU, does not reflect in a better overall cost effectiveness [8].

The impact of different instances has been analysed [7] using various parameters tracking the cost associated with running three different applications that cover a wide range of scientific fields.
In figure 1, results obtained by the three applications have been analysed in terms of cost and execution time. In terms of reference, the default instance type configurations can be found in the Appendix A section.

From Figure 1, the top diagram allows us to see which instances provide advanta-

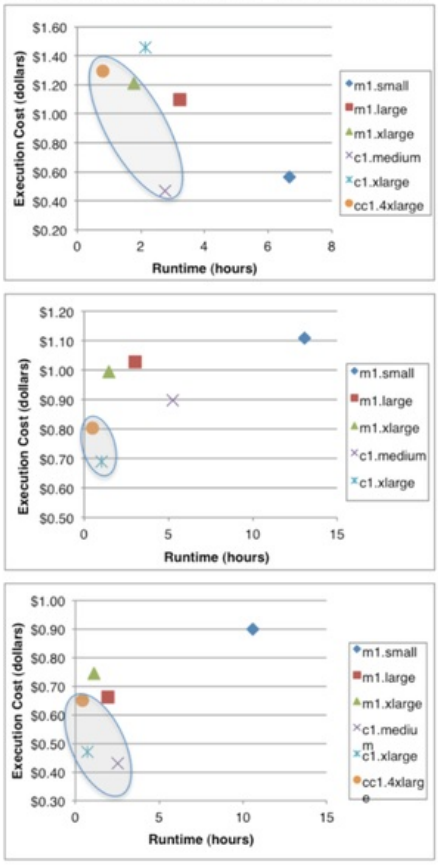


Figure 2.1: [7] Cost versus performance comparison for of three applications Montage, Broadband and Epigenome.

geous performance. For example it is clear that m1.large and c1.xlarge provide less optimal results despite the fact that its specifications are superior to c1.medium which shows better performance in terms of cost and similar execution speeds. Conversely, on middle diagram of Figure 1, the choices that performed well for the results obtained in the top diagram are not the same. In the middle diagram of Figure 1, it is clear that c1.xlarge has significantly lead performance when compared the results in the top diagram of Figure 1. On the third test results, the Epigenome application (bottom of Figure 1), shows a different Pareto optimality as circled by the blue set, which shows that c1.xlarge and c1.medium perform differently when compared with Montage and Broadband. From these results the Pareto efficiency effect demonstrates that better performance of one application through a particular instance has potentially worse performance when used in a different program. Furthermore, the study [7] showed that the run-time for different applications may not necessarily improve linearly despite the cost of hiring resources is significantly increased.

2.3 Hadoop ecosystem

Hadoop is a framework based platform developed to facilitate cloud-based applications on cloud services such as Amazon Web Services for processing large scale cluster based data processing [19].
Hadoop a open source implementation allowing large scale input data to be equally distributed into smaller fragments for easier computation. For this purpose, Hadoop file system HDFS (Hadoop Distributed File System) allows the possibility to store data on the available nodes provided by the system which may include Cloud based solutions such as Amazon Web Services.

The Hadoop platform utilizes the MapReduce system for safely processing large amounts of data. Hadoops' role further takes into consideration factors such as hardware failures by providing a recovery mechanisms.

2.3.1 HDFS

HDFS is the core of the Hadoop file system providing storage and transmission of large data through streams while utilizing the resources available on a commodity hardware in the cloud.

Large Input Files

HDFS provides the capability to store large amounts of data, that is in the range of megabytes and even files over several terabytes.

Streaming Data

HDFS revolves around the simplistic data processing patters that follow the idea of reading the data many times iteratively while writing the data only once. Once data is copied from the source, iterative computations are performed upon it.

Commodity Hardware Resources

The high availability of affordable ready-to-use cloud resource provide researcher and developers to hire large quantities of machines such as Amazon EC2 instances. Despite the availability of readily accessible computing resources, clusters can accommodate nodes from tens to thousands of machines [4] [11] [26]. Due to the nature of cloud resources, the location and durability of the machines especially in application that require long execution times can bring large clusters to a halt in the event of machines failing.

Despite of this fact, HDFS was designed in such a way that small resource failures do not interrupt the overall processing of applications. Thus, providing safety barriers for such events.

Data Latency Access

Data access latency is referred as the time required for data to be accessed by the computational unit from its original storage location.
The HDFS model focuses on throughput of data which may trade-off against latency. However, HDFS reduces the issue since each computing node views the data set distribution is aligned. In terms of low-latency scenarios, if the data distribution is not aligned, all the records in one table will be compared to all other tables which are located on separate computing nodes.

Large Fragments of Input File

As the amount of input fragments increases, namenodes of the cluster bear the responsibility in regards to storage of file-system metadeta within memory, thus the amount of files largely affects computational performance since the amount of memory on each namenode affects the number of files that can be stored.
These limitations that surround the architecture of HDFS may affect the volume of data that can be processed, however the limitations of HDFS may only start to show these pitfalls once files in the billions start being utilized [24].

2.4 Cluster Resource Scheduling

In order to process scalable graphs, depending on the size of graph data, numerous factors must be taken accounted for when utlizing specific scheduling algorithms to optimally assign computing resources graph-processing frameworks. These factors include:

- Scalability
- Cost
- Execution Time
- Graph Size

In detail, processing application/frameworks are unlikely to be linearly scalable. Thus, the resource scheduling algorithm must take into consideration algorithmic scalability, cost and resource capacities. Hadoop based clusters utilize several resource allocation platforms. YARN, is one of them most widely used resource negotiator for distributing shared resources within a cluster environment. Currently, Yarn supports several resource scheduling algorithms. Allocating Amazon EC2 computing resources depends on various factors listed above [14] [1]. Thus, scheduling algorithms of these resources must consider the following properties of scheduling algorithms:

- Deadline Based
- Budget Constrained
- Execution Time
- Time Constrained

2.4.1 FIFO Queue Based Scheduling

Most frameworks such as the Hadoop ecosystem, utilize First In First Out (FIFO) scheduling which can provide high utilization in a cluster environment which can result in long response times. This is particularly true when the amount of jobs to be executed is outweighing the available computing resources [2]. Since larger jobs can take longer to complete shorter jobs, the shorter job will have to wait in queue for the longer job to complete. An alternative scheduling solution which was proposed by Facebook called the Hadoop Fair Scheduler(HFS) [20] [16]. Its aim is to provide improved data locality by calculating the fair share for each job/task. This technique would reduce the overall time of graph processing especially critical for large graphs.

2.4.2 Fair Scheduling

Fair Scheduling technique is designed to allocate resources to jobs in a way that overall all jobs receive a uniform portion of resources over a period of time. This design allows queues to be arranged in a hierarchical way for dividing resources. The Fair Scheduler utilizes a queue path which initially allocated resources equally to each queue. Fair Scheduling [10] guarantees the minimum allocation for queues, however if the resources allocated to a queue are not being fully used, they are then reallocated.

2.4.3 Express Lane Scheduling

Express lane scheduling can be utilized in conjunction with the Fair Scheduling algorithm by providing special considerations to short tasks when all applications are occupied, in order to provide higher priority to shorter tasks.

Chapter 3

Apache Giraph

In this chapter Apache Giraph framework is introduced through recent literature reviews as well as an overview of the system model.

3.1 Overview

Apache Giraph is a distributed graph processing framework adapted from the Pregel graph processing ideology innovated by Google. Giraph applications are designed to run on top of the Hadooop ecosystem, which provides a base framework platform for developing distributed graph processing algorithms.
Apache Giraph utilizes the bulk-synchronous methodology in terms of graphs where verticies communicate with adjacent-connected neighbouring verticies during every superstep. Each superstep consists of a barrier mechanism where all running workers must complete their computations prior to the initialization of the next superstep, thus ensuring uniform progression for each superstep computation.
Giraph further utilizes fault-tolerance capabilities to deal with failures in large cluster environments. Check-pointing is used for achieving fault-tolerance where the master of the cluster instructs all workers and detects failures which allows the master to terminated failed workers.

3.2 System Model

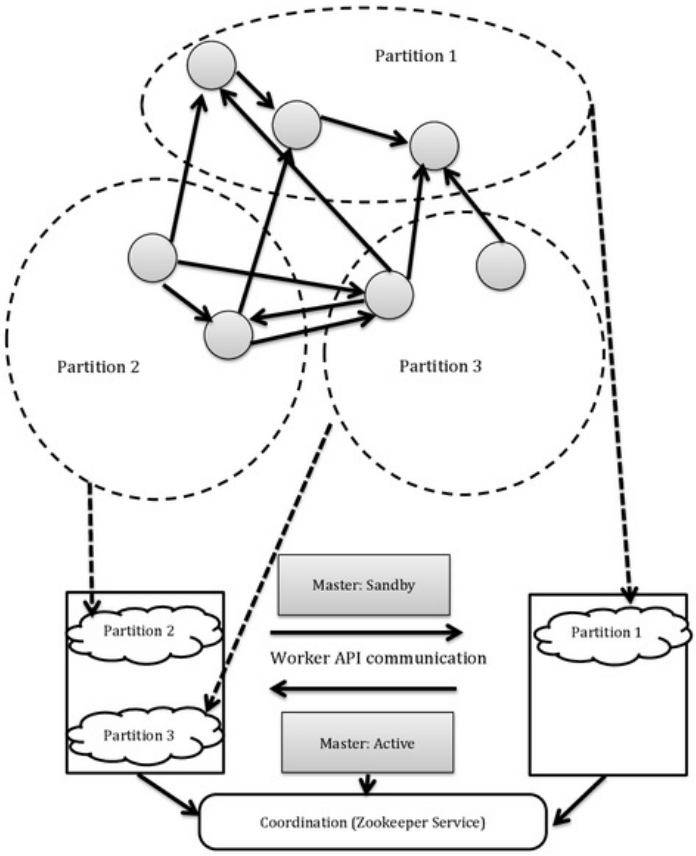


Figure 3.1: Partition System Model of Apache Giraph

3.3 Giraph Concepts

3.3.1 Master

For each running Giraph application, a master node is focused on computational coordination for the worker nodes. Once a master node becomes active, the tasks that it performs include synchronizing all the workers collectively for each superstep.
As each superstep progresses, the master initializes the workers by partitioning graph data to the available worker nodes through the use of a graph partitioning methodology. Further, the master node is responsible for executing the compute function code as well as monitoring the health and status of workers present in the same cluster environment.

3.3.2 Worker

Individual workers are the responsible computing units for processing the graph components that have been specifically assigned to a worker by the master. As the application progresses through each superstep, the worker is responsible for the execution of the compute method for the graph component that has been assigned to that particular worker. As Giraph uses a fault-tolerate based technique, each worker is responsible for communicating with the master to ensure that failed workers are terminated from the application rather than being carried throughout.

3.3.3 Coordinator

Coordination services are outside the scope of graph processing services such as master and workers. Coordination services such as Apache Zookeeper. Each Zookeeper ensemble is a representation of nodes running a coordination service while providing synchronization and fault-tolerant configurations necessary for distributed applications. Zookeeper serves important roles as it provides persistent distribution between node ensuring that a failed node will be replaced by a healthy available node. This scenario is evident when choosing a master node, where a chosen node can be re-assigned to another node if the currently selected node fails.

Chapter 4

Graph Drawing Algorithms

In this chapter graph drawing algorithms are introduced through recent literature reviews where concepts in this chapter are further investigated and experiments are performed in the following chapters.

4.1 Background

Initial frameworks such as MapReduce [**?**] provide a simple programming model for building scalable parallel algorithms to process large amounts of data on clusters. However new frameworks have improved the need to design iterative jobs manually by chaining multiple MapReduce tasks, as well as Hadoop extensions such as HaLoop [15], Twister [**?**] were introduced to optimize the iterative design of MapReduce. Graph processing platforms such as Pregel, Giraph, Mizan, GPS and Pregilx are based on the (BSP) Bulk Synchronous Parallel system that uses a vertex-centric programming model with computations on vertices being represented as sequences of supersteps. Pregel system was introduced by Google it's said to be the first BSP implementation specifically for programming graph algorithms using "think like a vertex" paradigm.

Further systems have been developed to exploit the optimization benefits of Pregel. Apache Giraph was implemented in Java to work on top of the infrastructure of Hadoop framework. Giraph is designed to run processing jobs as map-only jobs on Hadoop and utilizes the Hadoop HDFS for data input and output.

BSP based systems such as Apache Hana focuses not only on graph processing but also other applications such as algebra and matrix computations. Other systems such as GraphLab [9] implemented using C++ which differently from Pregel and Giraph and GraphLab depends on shared memory abstraction and the (Gather, Apply, Scatter) processing model [23]. Several graph processing platforms such as GraphX, GraphChi, PowerGraph all are suitable platforms for conducting further research into graph drawing.

4.1.1 Fruchterman-Reingold: Force Directed Algorithm

Force-directed algorithms have evolved in their design and implementation strategies have been developed over the years. Fruchterman-Reingold [25] [6] follows the "spring-embedder" model allowing the drawing of graphs consisting of straight edges where the uniformity of the length of edges must be consistent. Thus, allowing designs such as the [28] to follow the forces of a natural system where vertices should not be drawn too closely in order to produce graphs in an aesthetically pleasing manner.
The main criteria that Fruchterman-Reingold algorithm follows in terms of aestheticism is to address the following design points.

- Spread the verticies uniformly across the visible space.
- Reduce the amount of edge crossings for each vertex.
- homogeneous lengths of edges, where each edge is scaled appropriately.
- Accommodate drawings to single frames.

The term "forces of nature" represent the design model where edges are replaced with strings and vertices are represented by rings which are located in a initial position in the overall graph. Thus, the forces of each spring determine the the new position of each vertex in its final state.

Despite of several force directed algorithms following a stricter system of laws of physics such as implementations based on Hooke's law [13], several force-directed implementations including Fruchterman-Reingold differ in the computation of attractive forces which are only computed in terms of locality, that includes only the attractive forces between the neighbouring vertices.

Chapter 5

Risk Assessment

Cloud computing aids in the delivery of efficient computing services as a cost effective solution for on-demand resource requirements. Cloud resources offer a great economical model for data processing and information handling offering highly attractive solutions.
Despite a shift towards cloud technologies due to affordability and low start-up requirements, leaning towards this solution surfaces potential risk that if undermined can consequently create colossal damages financially and in terms of information security.
In this chapter, we elicit potential risks associated with cloud computing, specifically relating to our research project then further in the chapter mitigation strategies for dealing with the associated risks will be developed.

5.1 Risk Rating

In determining risks, we follow ISO 31000 risk management as a universal methodology for implementing risk assessment. For risk classification, a risk assessment matrix developed by the Australian Government - Information Security is utilized as part of risk rating. the guidelines followed in this chapter as shown in Figure 5.1

LIKELIHOOD	CONSEQUENCES				
	Insignificant	Negligible	Moderate	Major	Extreme
Remote	VERY LOW	VERY LOW	LOW	MEDIUM	HIGH
Unlikely	VERY LOW	LOW	MEDIUM	MEDIUM	HIGH
Possible	VERY LOW	LOW	MEDIUM	HIGH	VERY HIGH
Likely	VERY LOW	LOW	MEDIUM	HIGH	VERY HIGH
Almost certain	LOW	LOW	MEDIUM	HIGH	VERY HIGH

Figure 5.1: Risk Assessment Matrix

In reference to Figure 5.1, the risk matrix likelihood of a risk occurring is a combination of the consequences and the probability of a specific risk from occurring. Risk are to be labelled as Very Low, Low, Medium, High or Very High based on the severity.

5.2 Risk Register Plan

In this section, we elicit potential risks involved with cloud computing. Based on each risk, IOS 31000 risk management methodologies are employed for categorizing each risk based on its characteristics.

Table 5.1: A sample long table.

ID	Name	Description	Probability	Impact
R1	Provider Lock-in	Using a single cloud provider can create difficulties migrating from one provider to another	Possible	Major
R2	Resource sharing	Sharing of resources can open potential to malicious activities by other parties accessing the same resources	Possible	Extreme
R3	user access	Having more than one user access the same resources can jeopardize security if one of the users does not follow safety procedures	Possible	Moderate

Continued on next page

Table 5.1 – continued from previous page

ID	Name	Description	Probability	Impact
R4	Security Groups	Lack of security groups/rules can allow unwanted users intercept communication within resources by accessing them	Possible	Extreme
R5	Access keys	Loss of private keys if used for accessing resources	Possible	Extreme
R6	Local storage	Unsecure storage of private keys can potentially give access to the attacker	Unlikely	Extreme
R7	Application breach	If access to a single application is breached, other applications sharing the same resources are potentially at risk	Possible	Extreme
R8	Resource termination	Auto termination features if failed, can allow resources to continue operations increasing unnecessary costs	Unlikely	Major
R9	Automation	If a user does not manually terminate a cluster or machine, monetary charges will be made despite of usage	Remote	Negligible
R10	Backups	Terminating clusters of resources can lead to data loss if no backups are available	Remote	Moderate
R11	Fixed cluster size	Fixing the amount of instances in a cluster can limit the potential of resource expansion based on resource demand instantly	Unlikely	Medium
R12	Backup loss	Loosing processed data backups will require re-testing the same data again, which can be costly depending on its size	Remote	Moderate
R13	Auto scaling	Resource auto-scaling if not carefully managed, can lead to over-resource purchasing increasing cost charges instantly	Remote	Major
R14	Updates	Multiple frameworks are utilized for performing data processing, if one framework is updated it can halt the application and redesigning of the algorithm may be required	Remote	Negligible
R15	Updates 2	Multiple frameworks are utilized for performing data processing, if one framework is updated results obtained can be inconsistent with previously collected data	Remote	Negligible

Continued on next page

Table 5.1 – continued from previous page

ID	Name	Description	Probability	Impact
R16	Cost increase	As demand grows, the cost of hiring cloud resources can dramatically increase especially for long term projects where changes are more difficult to perform	Remote	Moderate
R17	Internet access	Loss of internet access can leave data out of reach	Remote	Major
R18	Service outages	Cloud providers can perform service outages, leaving he project out of control in terms of data and access to resources	Unlikely	Extreme

5.3 Risk Mitigation Plan

Identifying risks and taking sequential steps to reducing the adversity of risks is a critical aspect towards project success. In section 5.2, we identified potential risks that may affect this research project or similar. Each risk has been categorised based on a risk assessment matrix in accordance to ISO 31000 and identifying each risks probability and potential impact on the project.
In this section, we utilize risk management strategies that best apply to each individual risk in order to reduce and prevent each risk from occurring. All of the risks identified in section 5.2 have been developed prior to commencing with the project. However, as the research project developed, more risks were identified and added to the Risk Register Plan section.
Four risk management strategies were are considered when applying risk mitigation plans for each risk. These strategies are as follows:

- Risk acceptance
- Risk avoidance
- Risk limitation
- Risk transference

In table 5.2, risk mitigation strategies are analysed based on each risk. Each risk identified in section 5.2 is referred to by its Risk ID.

Table 5.2: Risk Mitigation

Risk ID	Risk Name	Mitigation Plan
R1	Provider Lock-in	For long term projects, it is important to perform local experiments to reduce the need of hiring cloud resources where possible
R2	Resource sharing	Limiting cluster resources sharing so each user has access to instances that are not being shared.
R3	User access	Ensuring both/all users sharing the same resource follow the same security access procedure such as using key-pairs instead of password-less point of entry
R4	Security groups	Strict guidelines should be followed when setting security groups. Guidelines should be followed such as those provided by Amazon Web Services
R5	Access keys	Storing private access keys on secured reliable machines is critical. If the key is lost, access to resources will be denied and regaining access is impossible
R6	Local storage	The location where the key is stored within the local machine, must be kept within a secure directory where only authorised users can access
R7	Application breach	All applications being run on a cluster must follow the same guidelines during setup. That includes using separate security keys so if one application is breached, other applications are still inaccessible from the adversary
R8	Resource termination	Termination features within frameworks allow applications to terminates once a task is completed. If the application never completes, the user can overstep the budget. Thus, manual or automated solutions for ensuring applications do not exceed a specified time limit
R9	Automation	Each instance/cluster of resources must be terminated or stopped after it has completed computations. Further, automated alarm systems should be utilized to prevent this event
R10	Backups	After each run of a application, the processed data should be downloaded or stored in a safe cloud space such as Amazon S3 for later retrieval
R11	Auto scaling	Auto-scaling should be limited and upper-bounds should be set to avoid excessive usage beyond user predetermined settings

Continued on next page

Table 5.2 – continued from previous page

Risk ID	Risk Name	Mitigation Plan
R12	Backup loss	Storage of logs to obtain runtime and other crucial results from application completion should be stored in cloud or local based backup locations such as Amazon S3 or local storage
R13	Auto Scaling	Auto-scaling should be limited and upper-bounds should be set to avoid excessive usage beyond user predetermined settings
R14	Updates	Each application installation should use the same framework versions to prevent results data from being inaccurate. Using configuration files to determine versions is a first step to ensuring accurate installations are performed
R15	Updates2	Framework updates occur regularly, thus backing-up previous versions can ensure the application will perform accurately until the necessary modifications are performed for ensuring application compatibility
R16	Cost increase	If a project is expected to exceed over a span of years, considering other platforms including local/offline hardware setup should be taken into consideration
R17	Internet access	Implementing local installations of the frameworks and applications on local machines must be performed in the case of internet access loss
R18	Service outages	Local and other cloud providers should be considered in the case of the current provider has an outage of service

Chapter 6

Experiments

6.1 Introduction

Graph drawing algorithms have adapted towards improvements in processing capabilities and the way we process data. Platforms such as Amazon Web Services, Microsoft Azure and several cloud processing providers have expanded the scope of students and researchers to perform larger complex graph processing algorithms through the utilization of clustered processing units. These machines are readily available to the users, where machine specifications are highly customisable and optimizable for custom processing requirements.

Abundant resources available provide several benefits for graph drawing applications and more generally distributed processing algorithms. Benefits include low start-up times as the need for low level hardware configuration is reduced and faulty machinery is easily replaceable without having an impact on project time lines where time based constraints are enforced. Further, the cost associated with setting up the machines is relatively low and affordable especially for low-budget projects or research grants where the availability of funds is limited.
Despite of the mentioned benefits, drawbacks and resource wastage is a main concern among distributed applications and frameworks.

Resource leaks, or lack of optimization fine-tuning, can lead to increased project costs in terms of monetary budget as well as potentially elongated processing times particularly critical when large data sets are being processed. Fine tuning the way frameworks and algorithm specifically utilize the allocated resources, and through experimental observations of resource usage patterns, it is possible to obtain trends, from which data extracted accommodates formulation of models where consequent applications that perform in a certain pattern may be able to benefit from previously developed models.

In this chapter, experiments are conducted to observe resource usage patterns as well as a critical observation of the effect resource consumption patterns have on several applications where data size, type and structure may constantly change. Further, the overall cost effectiveness associated with distributed algorithms and its effect on project completion are investigated through multiple case studies.
Consequently, through fine tuning, optimization and formulating resource consumption patterns through experimental data, models and methodologies will be investigated and developed to further improve processing speeds and the overall effectiveness of large scale cloud based distributed resources.

6.1.1 Resource Allocation - Workers

Resource allocation patterns and optimization experiments conducted in this chapter are conducted through the use of Amazon EC2 machines called instances. Each instance is a physical machine which has specific resources allocated depending on the type of machine hired. The machine names and resource specifications are included in Appendix A section.
Throughout the experiments, we may refer to machines as 'workers'. Each worker consists of its individual resources and its considered as a processing unit. For every instance, it may be possible to have more than one worker, which is highly dependable on the number of CPU cores on each instance.

6.1.2 Instances - On-demand Pricing

Instances used in the experiments have a particular monetary cost associated with them. The costs depend on the current market price available at any given time. However, for consistency purposes all the costs for each instance used in the experiments are included in Appendix A.

6.1.3 Spot-Instance Pricing

Spot instances through Amazon Web Services allow price bidding which can reduce costs of each instance up to 80 % depending on the market forces at any given time. With increased price reductions, the risk of instance prices being out-bidden can reallocate the instance resources to the highest bidder causing running applications to fail in most cases. Thus, spot instances are generally avoided in the following experiments due to the volatility and the risk of application failure.

6.1.4 Instance Availability Regions

Prior to instances being purchased, it is possible to select between multiple geographical regions where machines are physically located. Each region has several effects on the hired instances, however, the main distinction being cost.
In all the experiments, the Sydney region instances were used and its prices are available in Appendix A.

6.2 Optimizing Worker Count

Amazon EC2 clusters facilitate the ability to hire instances of specific configurations. Graph processing frameworks such as Apache Giraph allow the specification of the amount of workers to be used for each job. A worker is a processing unit where its main function is to perform computational task on a graph partition through the compute() method.
Initially, each worker is assigned a fraction of the graph data to be processed. the master node which oversees the workers in the current job, assigns the input splits of the overall graph data. Once no inputs split remain unassigned, the master node signals each worker to continue into the next or initial superstep.

Depending on the graph partitioning strategy, each worker will be assigned with its specific input split. In theory, the higher the amount of workers, the higher parallel computations can occur. Regardless, several potential bottlenecks can occur in terms of memory competition, data locality and network communication between each worker.

Each superstep in a Giraph application represents a iteration of computations from all workers. More workers may be able to process more data simultaneously, however, that may not directly reflect in improved execution time of the job.

Individual supersteps are composed of several layers. Firstly, each worker works independently using its own share of cluster resources while computing the graph vertices and edges that are part of its own input split.

6.2.1 Worker Resource Scheduling

As demonstrated in Figure 4.1, each workers local computation behaves independently, thus, its execution speed may not necessarily be synchronized with other workers in the local cluster. In the case of resources being equally distributed

between workers correspondingly close execution times may result, especially for smaller graph data sets. Despite this fact, due to the percentage of graph edge locality, the execution time can be affected. This case theoretically also applies to workflows that include more that a single instance as Giraph is unaware of workers being in the same instance thus treating them similarly when they are distributed amongst different instances.
In this section of the experiment, we will discover the impact of multiple worker per instance model, by observing the execution time of a connected components algorithm part of a graph drawing application for Apache Giraph.

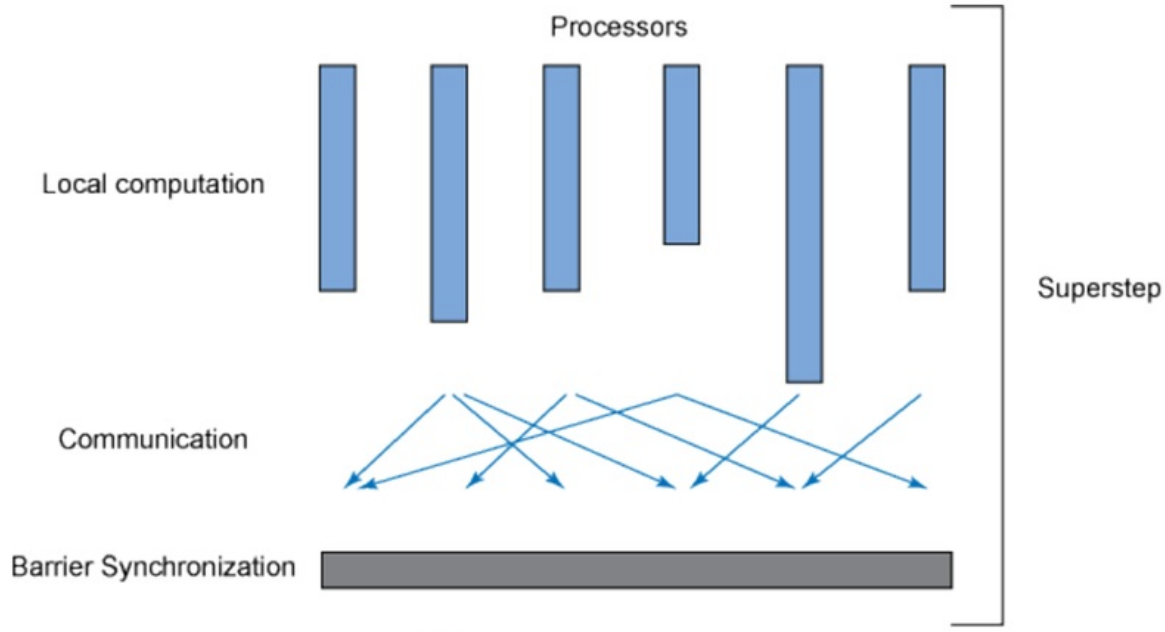


Figure 6.1: [21] BSP Workflow model

6.2.2 Graph Execution Flow

The Apache Giraph framework uses a synchronization barrier system to ensure all workers increment into the next graph processing iteration. The Barrier works by holding each worker from continuing into the next iteration until all workers in the job workflow have completed the execution as shown in Figure 4.1. After all vertices have voted to halt, thus signalling that no messages are left to be transferred. Prior to the continuation into the next superstep, all workers must have completed their

computation of all edges and vertices. The next superstep will initialize if-and-only-if all workers have communicated their superstep completion to the master node.

Despite of the Synchronization barrier serving as a system for synchronizing workers, depending on the graph locality and the level of graph input splits being balanced between each worker, not all workers may complete execution at the same time.
As the amount of workers increases per instance, the probability for individual workers to complete computation for each superstep equally in a measurement of time may increase the overall execution time. Specifically, having more than one worker per instance may increase the probability of workers completing their execution at different times versus a single worker per instance which would result in a single execution with no other workers potentially taking longer to complete.
Thus, in this section of the experiment we will investigate if the amount of workers bears any significance to the completion time of each worker within a single superstep of the job workflow.

6.2.3 Workflow Processing Usage

As workers are added to an instance, the CPU usage may be an indicator of resource usage and competition between workers of individual instances. During the scenario of multiple workers per instance being allocated with specific fragments of processing power, the overall CPU usage will indicate the total time of all workers combined spent on map and reduce tasks for the particular workflow.

6.3 Introduction - Resource Provisioning and Scheduling

Resource provisioning and allocating resources in distributed cloud environments correlates with the overall performance and cost of applications in clustered resources. As the number of machines utilized in a cluster rises, the likelihood of resource usage patterns increases. Moreover, the possibilities for optimizing resource sharing at application and task level presents potential for further improvement.

6.3.1 Resource Provisioning Model

Understanding the fundamental model of resource provisioning is an essential component towards optimization of graph processing algorithms where algorithmic optimization has been exhausted.
Running graph drawing applications through frameworks such as Apache Giraph, firstly requires several underlying frameworks and resource management frameworks allowing for efficient allocation and distribution of homogenized resources in a cluster.

In the experiments conducted throughout this chapter, Apache Hadoop Yarn (Yet Another Resource Negotiator) framework is used as a case study. The reasoning behind the chosen application is due to the wide variety of applications and the ability to support most of the latest scientific and graph processing frameworks. In Figure 5.2 an overall view of resource allocation is illustrated.

In the experiments, the aim is to generalize resource consumption patterns which are not specific to individual frameworks and methodologies. Consequently, the use of specific resource management and graph drawing frameworks are utilized as a case study for conducting the experiments.

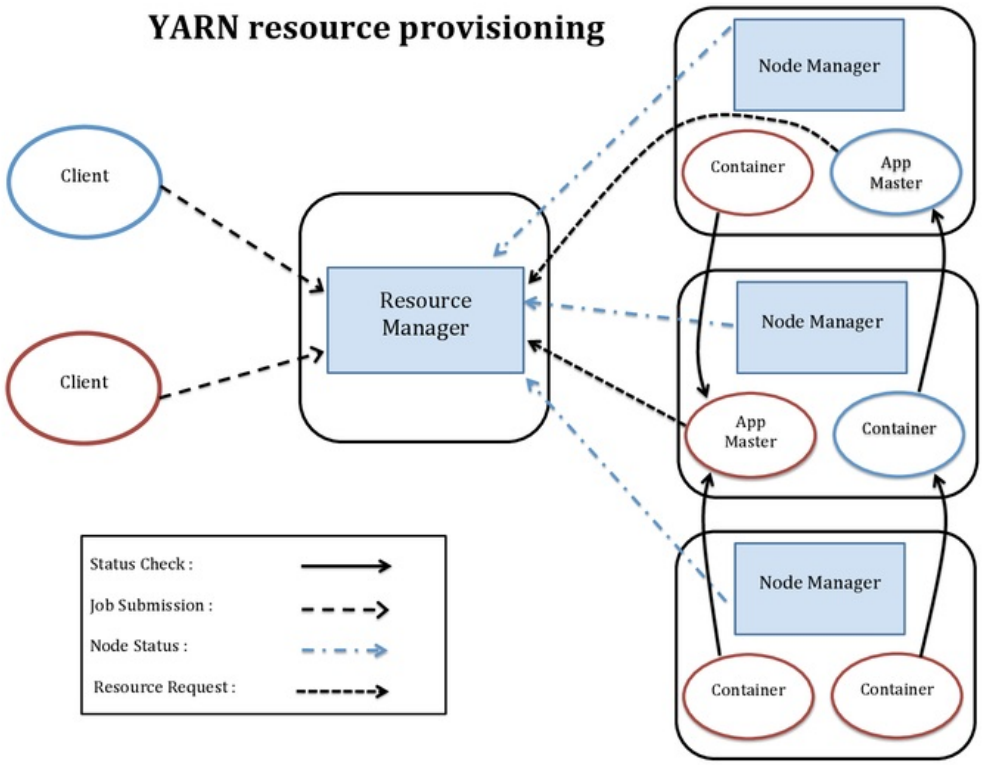


Figure 6.2: YARN resource provisioning model

Client

In Figure 5.2, the client represents the user that wants to submit a specific application. The user is able to submit the application directly to the Resource Manager.

Resource Manager

Resource Manager serves as an intermediary between the user and overall management of applications while managing the overall cluster resources.
It has access to all applications that are in queue, running and failed. Nonetheless, the Resource Manager's main objective is the management of cluster resources and determining the application that will receive a portion of cluster resources.
The logical computations performed to select the next application to receive a portion or all of the resources is determined through a scheduler. Consequently, based

on the scheduling methodology an application(s) is chosen.
The Resource Manager allows incoming resource requests if-and-only-if the application has been accepted. Succeeding application acceptance, the following step for the Resource Manager grants or denies resources for the accepted applications with the Application Master.

Application Master

The role of Application Master is to perform resource requests and perform negotiations with the Resource Master. Each individual request is composed of requirements such as the following:

- Specific quantities of CPU cores
- Amount of memory
- Hardware rack-name/location (depending on configuration).

Concluding successful resource provisioning between Application Master and Resource Manager, the Application Master requests the Node Manager to initiate application tasks. Further roles performed by the Application master include:

- Continuous resource negotiation
- Local resource allocation
- Management of task scheduling
- Task execution.

Node Manager

The focal responsibility of Node Manager is to initiate application tasks through the utilization of resources available within the container. The tasks run by Node Manager are not framework specific, however in our experiments application tasks are represented by graph processing tasks within the Apache Giraph framework. Conclusively, Node Manager observes and tracks resource usage for individual containers and terminates containers if resource consumption exceeds the specified limit or in the case of a user request to terminate a container.

6.4 Resource Provisioning Models

6.4.1 Introduction

Efficient Resource provisioning is of significant importance in relation to utilizing the available resources to accomplish maximized return on investment. As cloud resources become evidently abundant, under-allocation and over allocation equal to wastage in terms of monetary cost, computational times and can impact successful completion of a specific application.

The following attributes are the essential components required to form conclusive optimization improvements towards resource provisioning:

- Memory available in a cluster of instances
- Lower bound resource requirements
- Size of data
- Data attributes
- Cost constraints
- Time constraints
- Memory allocated for resource management.

6.4.2 Objective

In this experiment, the aim is to observe resource consumption patterns for graph drawing algorithms and consequently derive a mathematical model for approximating an lower-bound resource usage such that the application is able to complete its computation whilst using an approximated minimum amount of resources.
Due to the nature of graph drawing algorithms, the foremost resource usage that is to be analysed is memory. The derived models should be applicable to similar frameworks that may have similar trends in resource consumption patterns.
Firstly, a broad based model is to be produced with the aim of increasing the scope of its applicability to graph processing and scientific applications. Furthermore, as a case study, experiments will be conducted on a graph drawing framework/application called Gila which is a Apache Giraph framework based application, while YARN will be employed as part of the cluster resource management.

6.4.3 Tools Required

In order to conduct this experiment, the following tools and data will be used.

- Access to Amazon Web Services
- Sufficient funds to purchase a cluster of 1 to 20 instances of r3.Xlarge instance type

  The cost of r3.Xlarge can be found in Appendix A1. Note: The cost of instances may change however the pricing used in the following experiments has been included in Appendix A1
- Graph data sets with the following properties

  Graphs with different amounts of vertices and edges

  Graphs of different composure, example synthetic and real
- 20 r3.Xlarge Memory optimized instances
- Access to open source frameworks such as:

  Apache Giraph

  Gila

  Hadoop

  YARN

  Git

  Apache ZooKeeper

  Maven

6.4.4 Procedure

One of main occurring issues associated with running graph drawing algorithms is determining the lower bound of memory required for the task to complete successfully. As part of the experiment, graph drawing applications through Apache Giraph and Gila are to be tested by recording the minimum amount of resources required for task completion. Ensuing, the test is to be replicated across multiple cluster sizes combined with numerous graph data of a range starting from 10,000 edges. Conclusively, from the collected data on graph resource consumption patterns, a model is to be developed for approximating the lower bound of resource requirements.

6.5 Pareto Optimality

6.5.1 Introduction - Pareto Optimality

Pareto Optimality is the notion of resource allocation where two parties or items cannot benefit one entity without making another entity worse off [18]. The notion of Pareto optimality is to assess multiple available options where selecting to improve one option it will lead to the worsening of another.

6.5.2 Introduction - Weak Pareto Optimality

In the context of cloud computing, having two conflicting entities where there is no such scenario or possibility that both cases are better off without another entity being worse-off can benefit from Pareto optimality analysis [17]. Thus, in the case of conflicting entities, it may be possible to improve one entity by a specified ratio while another entity is worse off by a lower ratio which can be considered as Pareto improvement.
Pareto efficiency does not factor fairness where depending on the user choice, if a desired item can gain at the loss of another, it can be considered as a Pareto improvement.

6.5.3 Introduction - Pareto Frontier

Restricting the number of possible options to only Pareto sets of choices allows the user or entity to shift between Pareto efficient options where a trade-off can be made within [12]. Pareto optimal solutions can also be categorized as Pareto Frontier solutions.

6.5.4 Performance vs. Cost

In terms of distributed cloud computing, Pareto efficiency can play a crucial role when evaluating options and trade-offs between performance and cost. In most cases, it may not be possible to have an improvement of performance and cost without any loss. Thus, if both entities cannot gain without any trade-off, Pareto optimality can be a valuable tool for making an optimal decision based on user needs.
Values included in the Pareto Frontier include points where it is not strictly dominated by another point. Points not included in the Pareto Frontier are set of points where they are strictly dominated by one or more points. Example:

$$f1(A) < f(C) < f2(A). \tag{6.1}$$

Where: - f1 is cost.
- f2 is performance.

Chapter 7

Results

In Chapter 4, experimental analysis of computational optimality of resource allocation is proposed for testing. In this chapter, experiment results will be analysed. The experiments were undertaken using a cluster setup through Amazon Web Services.

7.1 Computational Units

In this section, we display the results obtained from analysing cluster and algorithmic behaviour. As a case study, connected components algorithm is analysed as part of a graph drawing algorithm GiLA implemented in the Apache Giraph framework.

7.1.1 Cluster Performance

In figure 5.1, we test running the algorithm on a graph data set "add32" which has the following graph properties:

- Amount of vertices 4,960
- Amount of edges 9,462

The cluster configuration as a benchmark includes one master node and a core node which facilitates the workers. As the number of workers is increased, the execution time increased linearly. Due to the nature of Apache Giraph, each instance mapper can be utilized as a separate worker, where each worker is assigned an input partition for computation.
Results show that the amount of workers increases linearly as we add

more workers. In the experiment, a m3.xlarge instance was utilized for both master and compute nodes.

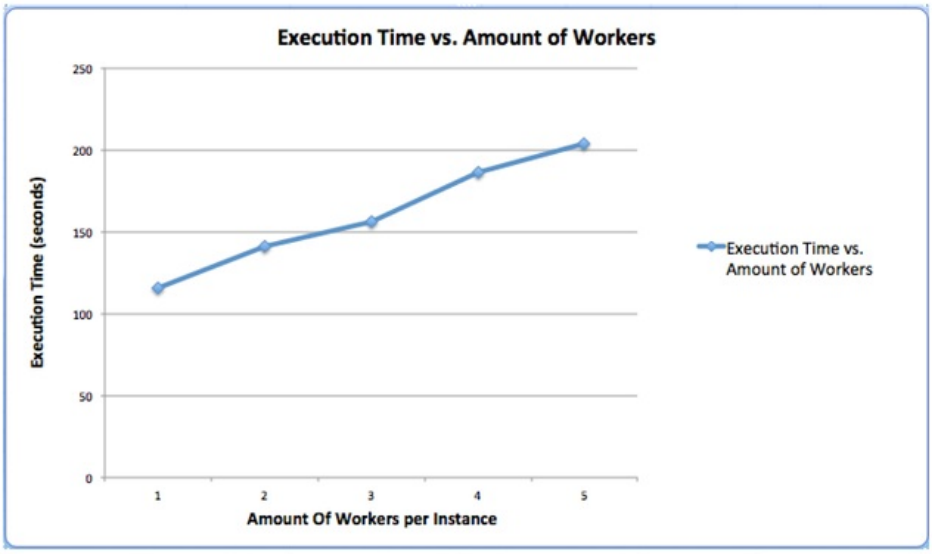


Figure 7.1: Computational Performance of Compute Units

7.1.2 Graph Locality

During the experiment, we gathered information in regards to graph partitioning locality. In this context, we refer to graph locality as the percentage of edges that are local to a worker as shown in Figure 5.2. As input data is partitioned, each worker is assigned a fraction of the graph. Vertices located in worker 1, may have incoming/outgoing edges that are connected to a vertex located on any worker (2, n-1] , where n is the amount of workers available on a single core instance. Consequently, more messages are required to be transmitted through the network.

The argument stays true when referring to a single instance rather than adding additional workers to multiple instances. Since more workers per instance increases additional resource competition, the underlying reason is due to resource of a instance being fragmented as each worker requires its own JVM, further increasing overall execution times.

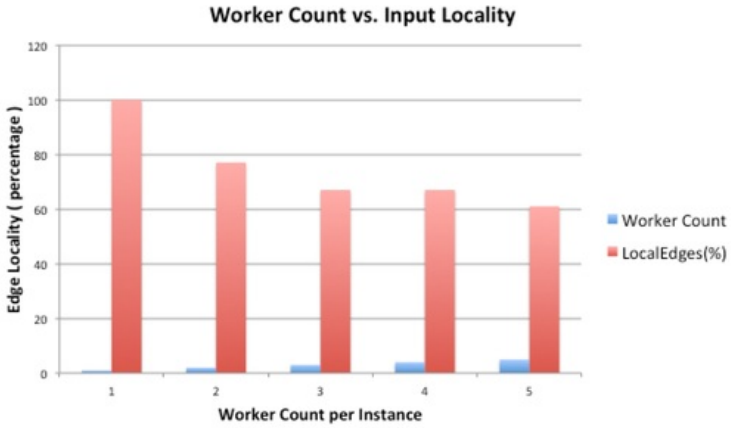


Figure 7.2: Edge Locality as the number of workers increase

7.1.3 CPU Usage and Written Data

More workers per instance increases the chance of delays since all workers must complete their computations before going to the next superstep.

From our analysis, the most optimal worker count in terms of CPU execution times and number of bytes written is to use a single worker per instance.

- In diagram 5.3, CPU usage increases in synchrony with rise in number of workers
- CPU times increases since more time is required for processing as well as writing files to local system, thus the amount of workers impacts the bytes written amount super-linearly
- Performance improvements while the number of workers is one worker per instance
- Increasing workers per instance, requires more JVMs to be launched, hence utilizing necessary heap memory.

Still, adding more instances improves execution time.

Figure 5.4 exhibits the bytes written in proportion to the amount of workers per instance. As the worker count per instance increases, resources will be exploited for writing data to local files and each workers resources are not allocated optimally.

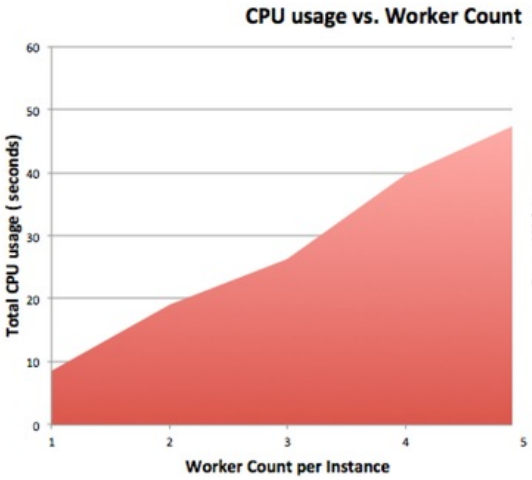


Figure 7.3: CPU usage as worker count is increased

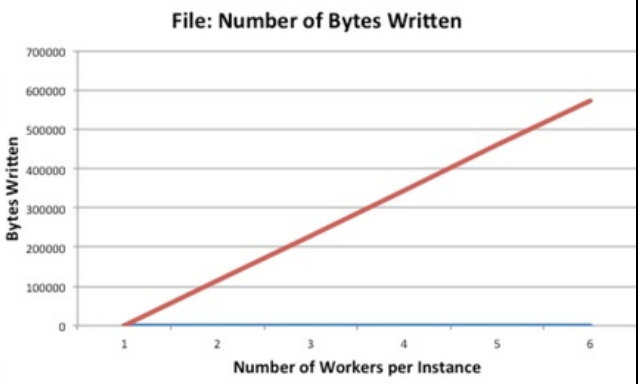


Figure 7.4: Bytes written as workers are increased per instance

7.1.4 Synchronization Barrier Delay

Increased workers per instance increases the chance of delays since all workers must complete their computations before going to the next superstep.
On a test case, we were able to observe that despite a balanced input split, the workers were initialized at the same time. Even so, their finish time varied with minor differences on a single superstep as show in Table 5.1. The test performed on table 1, includes a cluster with a master node and a single node. Five workers were instantiated simultaneously. As shown, worker 2 took longer to complete execution which results on the other workers waiting at the synchronization barrier as shown in Figure 4.1 before any of the workers can proceed to the next superstep.

Instance Used	Worker ID	Seconds To Complete Execution
m3.xlarge	0	18
m3.xlarge	1	18
m3.xlarge	2	23
m3.xlarge	3	18
m3.xlarge	4	18

Table 7.1: Worker run-time for a single superstep

7.2 Resource Provisioning Models

Resource provisioning and obtaining a lower bound of the resources required for running a distributed graph algorithm on a homogeneous resource cluster, involves multiple stages of computations before specific amounts of resources are determined.
In order to allocate a specific resource portion for a job/task, the following pivotal characteristics must be taken into consideration:

- Total memory available in a cluster, that is the combined memory from each machine hired combined
- The lower bound resource requirements for a task to complete successfully
- Size of the data being used
- Attributes of data being tested. Example, synthetic random graphs, real graphs, social graphs, network graphs or any other specific type of data being used. NOTE: Data can apply to scientific workflow tasks or similar data used in scientific workflow algorithms
- Cost based constraints
- Deadline based constraints, referring to the time it takes for a job to finish.

7.2.1 Container Allocation per Instance

The number of containers that can be allocated to a specific instance is dependable on several factors. Firstly, the amount of memory available, number of CPU cores and storage can impact the number of containers that can be allocated per instance. Depending on the mentioned factors, it is possible to determine the maximum amount of containers that can be allocated per instance. Conclusively, each container functions as a “worker” meaning that its main role in graph processing jobs is to perform computations. Hence, a container can be referred to more generally as a “worker”.

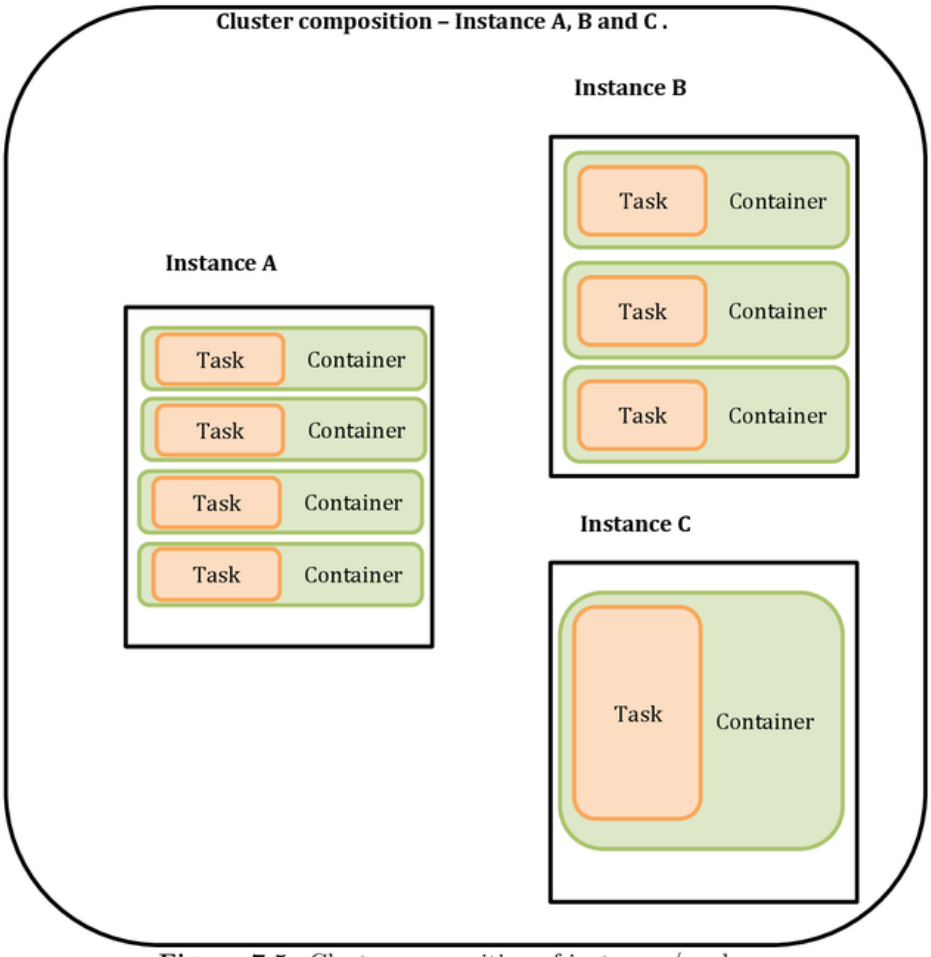


Figure 7.5: Cluster composition of instances/workers

As demonstrated in Figure 6.5, it is assumed that instance A, B and C are of identical resource specifications. However, instance A, B and C have multiple configurations where instance C has one container/worker. Conversely, instance A is composed of four smaller sized containers. Depending on the composition of the algorithm, it may be beneficial to use a single worker/container per instance, however increasing the number of containers/workers reduces the computations required at the cost of reduce portions of computing resources.

Thus, finding the maximum amount of containers/workers per cluster can be found using the following model representing a broader general use. Further in this chapter we test this model against a graph drawing application as a case study.

$$W_{max} = min[C \times 2, \frac{R}{M_{min}}] \tag{7.1}$$

Where:

- W is the maximum number of workers/containers that can be allocated in a cluster.
- M_{min} is the lower bound amount of memory to be allocated for each container/worker in a cluster. Note: each data set is likely to require different minimum amounts of memory, specifically of importance with algorithms that are memory intensive. The lower bound is calculated for the case study in the next subsection.
- C is the total amount of cores available in the cluster.
- R is the ram available in the cluster. R is calculate using the following equation:

$$R = I \times r \tag{7.2}$$

Where:

- I is the number of instances in the cluster.
- r is the amount of memory available in each instance.

7.2.2 Case Study: Lower Bound Memory Consumption Pattern

Uncovering the data resource footprint widens the scope of potential resource savings by utilizing a minimal portion of resources within a cluster. Through the analysis of the lower bound memory allocation pattern, in most cases a pattern is likely to emerge allowing consequent applications to pre-set the amount of resources for each data set based on its attributes and characteristics.

Running graphs of sizes from 10,000 edges up to 2,000,000 edges, resource consumption patterns start to emerge. From the data, mapping the results to an equation that allows potentially predicting the lower bound of memory requirement. However, the benefits of reducing resource usage increases completion time. Detailed, the advantages and disadvantages of using lower bound memory are as follows:

Advantages:

- A significant reduction of resource usage
- Freeing up resource portions for other applications or scientific workflows
- Allows concurrent processing of multiple data sets. Example: drawing multiple graphs
- Ability to launch and successfully complete jobs with minimal resource requirements.

Disadvantages:

- Completion time increases. Not optimal for time constrained based jobs/tasks
- Despite lower resource consumption patterns, overall cost increases.

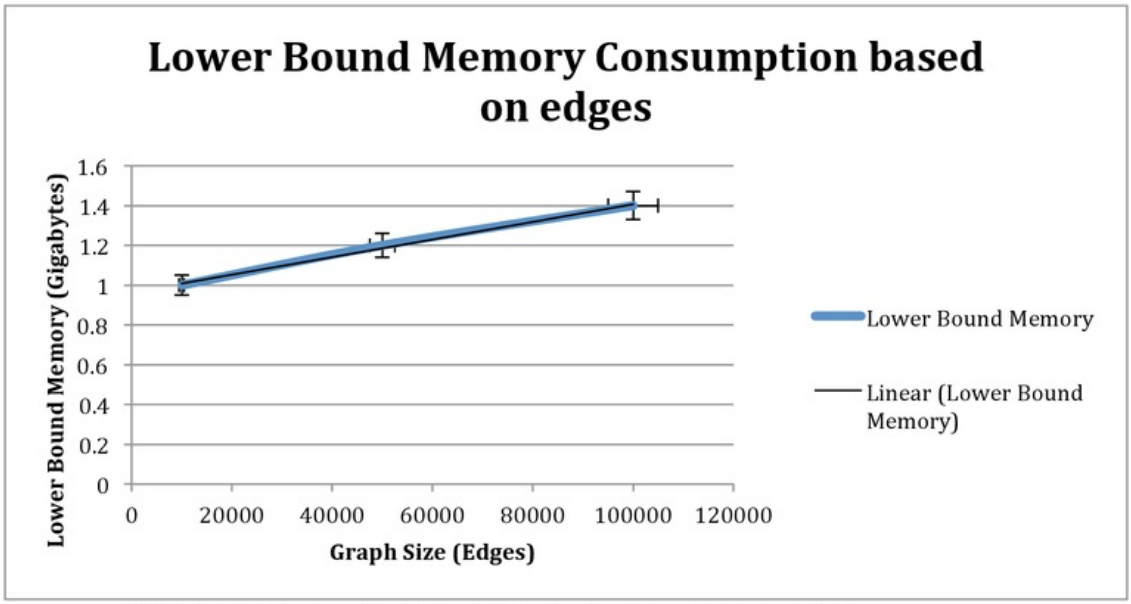


Figure 7.6: Lower Bound resource consumption based on edges.

As shown in Figure 6.6, it is observed that a pattern emerges on the minimum memory requirement for a job to complete successfully. The graph data tested consists of synthetic randomly generated graphs of the same degree.
Despite the pattern, the identical procedure was undertaken and we compared the results of similar as well as the same graphs based on the amount of vertices. The properties of the instances used in this experiment are as follows:

Model	CPUv	Memory(GB)	Storage SSD(GB)
r3-Xlarge	0	18	1x32

Table 7.2: Lower bound test: Instance specifications.

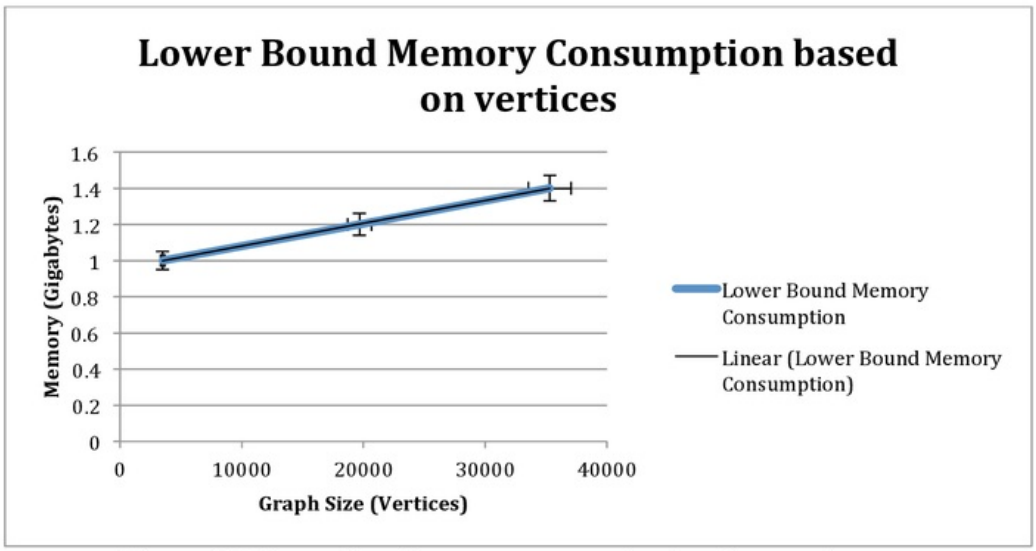


Figure 7.7: Lower Bound resource consumption based on vertices.

In Figure 6.7, the results of graph drawing resource consumption patterns are compared to the results based on Figure 6.6. From the results, the use of edges and verticies as a data characteristic shows similar results.
Based on the following data, the formulated representation of the results can be formulated by the following equation:

$$M_{min} = 4E - 6 \times X + 1 \tag{7.3}$$

Where:

- M_{min} is the lower bound amount of memory to be allocated for each container/worker in a cluster. Note: each data set is likely to require different minimum amounts of memory, specifically of importance with algorithms that are memory intensive. The lower bound is calculated for the case study in the next subsection.
- X is the amount of edges in the graph data set.
- E is the variable exponent.

Computing the maximum amount of containers

The model for computing the maximum amount of containers/workers was generally introduced by the equation 6.1.

$$W_{max} = min[C \times 2, \frac{R}{M_{min}}] \tag{7.4}$$

After computing the lower bound of memory required to successfully perform a graph drawing task, it is now possible to calculate the maximum potential amount of containers/workers. Below in table 1.1, several data points are calculated and compared to the results obtained by testing synthetic graphs using equation 6.3 in order to compute the variable M_{min} .

Graph Size(Edges)	Approximated minimum amount	Actual amount required
10,000	1.04	1
50,000	1.2	1.2
100,000	1.4	1.4

Table 7.3: Lower bound test: Instance specifications.

Following the computation of M_{min}, it is possible to calculate the maximum amount of workers possible using the equation in 6.4.

7.3 Minimum Amount of Workers Model

In the section 6.2, the maximum amount of containers that is feasible within each instance of a cluster is computed while concurrently incorporating the lower bound resource consumption. In this section, results and a model is introduced for computing the minimum amount of workers required to complete a job/task. Finding the minimum amount of workers required allows portions of a resource cluster to be freed up for use by other applications such as scientific workflows or running several concurrent graph processing applications.
A common trend amongst cluster environments is accessibility. A user who may be sharing a cluster environment has to take into consideration possible portions of resources being pre-allocated. Factors such as the following need to be taken into factor:

- Pre-allocated resources, example: Memory

- Total available resources including:
  - vCPU
  - Memory
  - Storage
  - I/O

Below a model is suggested for computing the minimum amount of workers required for a job/task to complete successfully.

$$W_{min} = \frac{R - S}{M_{min}} \tag{7.5}$$

Where:

- R is the ram available in the cluster. R is calculate using the following equation:

$$R = I \times r \tag{7.6}$$

Where:

- I is the number of instances in the cluster.
- r is the amount of memory available in each instance.
- S is the the amount of memory pre-allocated within the cluster.
- M_{min} is the lower bound amount of memory to be allocated for each container/worker in a cluster. The computation of M_{min} is demonstrated in section 6.2.2 in this chapter.

7.4 Maximum Resource Provisioning

Utilizing the minimum portion of resources available presents numerous benefits such as the ability to share cluster resource with other concurrently running applications. However, in the scenario where cluster resources are solely dedicated to a single individual application, then it is necessary to determine the maximum amount of resources that can be allocated towards an application.

Having the ability to determine the minimum amount as shown in the previous section, and, calculating the maximum amount of

resources that can be allocated, can help the user to quickly determine if the application will be able to run using available resources. In order to allocate the maximum amount of memory available, the following equation can be used.

$$M_{max} = max[M_{min}, \frac{R}{W_{max}}] \tag{7.7}$$

Where:

- M_{max} the maximum amount of memory that can be allocated for each container.
- M_{min} is the lower bound amount of memory to be allocated for each container/worker in a cluster. The computation of M_{min} is demonstrated in section 6.2.2 in this chapter.
- W_{max} is the maximum amount of containers/workers.
- R is the ram available in the cluster. R is calculate using the following equation:

$$R = I \times r \tag{7.8}$$

Where:

- I is the number of instances in the cluster.
- r is the amount of memory available in each instance.

7.5 Pareto Optimality

Pareto optimality facilitates clear trade-off consequences based on two entities such as cost and performance in-order to select priority-based options where one entity might be preferred over another dependant user scenario.

7.5.1 Cost Distribution - Pareto Chart

To evaluate efficient instance types for utilization in a cluster, it is paramount that a cost comparison between multiple instance choices is conducted. Firstly, the cost distribution between instance types is analysed. In the experiments conducted, only memory optimized instance types were used. Due to the nature of graph drawing applications being memory intensive in our experiments. In this experiment, the results shown in Figure 6.8 focus solely on r3 type instances as they optimized in terms of memory. Despite

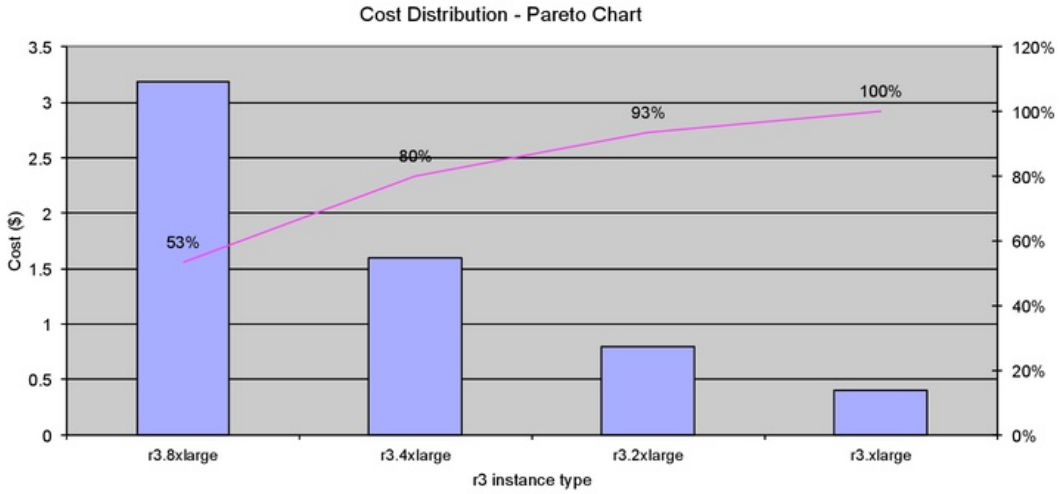


Figure 7.8: Cost only distribution - Pareto chart.

the linear equal distribution in terms of cost singly does not directly reflect a correlation in terms of performance output.
The ratio and relationship between cost and performance must be weighted as two unrelated entities in order to display the relationship between cost and performance in order to analyse and perform trade-off within the Pareto Frontier.

Model	CPUv	Memory(GB)	Storage SSD(GB)
r3-large	2	15.25	1x32
r3-xlarge	4	30.5	1x80
r3-2xlarge	8	61	1x160
r3-4xlarge	16	122	1x320
r3-8xlarge	32	244	2x320

Table 7.4: Lower bound test: R3 Instance specifications.

The cause of a linear distribution of cost for r3 instances is justified by the resources quantities provided by each instance as shown in Table 6.4. In this occasion, r3 instances follow a clear trend in cost to memory provided ratio, meaning that the price for r3.2xlarge amounts to 50% of that offered by r3.4xlarge.

7.5.2 Application Cost - Graph Drawing

In this section, we consider an application to be a set of data A where each data set consists of edges and vertices. Further, we consider multiple resource types 'i' within the Amazon Web Services where each resource type offers a different price for the amount of resources it provides. The amount of memory is represented by:

$< r_1, r_2, r_3, \ldots , r_i >$

The cost associated to each resource type is represented by:
$< c_1, c_2, c_3, \ldots , c_i >$
The resource considered is memory for simplicity purposes and the framework used as a case study, GILA and a graph drawing algorithm that is memory intensive in its nature. However, the resource is interchangeable and can be applied to other frameworks and algorithms in a cloud resource distributed environment.
In our results, only homogeneous resource clusters were considered where the compute workers/containers are allocated with equal resource quantities, especially necessary for the case study due to Apache Giraphs framework synchronization barrier favouring homogeneous distributions as demonstrated in section 6.1.4. The resources utilized in the results consider cloud based resources, namely Amazon EC2 on-demand instances.

7.6 Resource Computation Model

From the experiments conduced it is assumed that for a application A consisting of supersteps α (individual compute tasks) each supersteps within the application.
Each individual superstep consists of a completion time ratio T_α denoted by the following:

$$T_\alpha = \frac{T_{total}/T_k}{r_m} \tag{7.9}$$

Where:

- T_α Is the average completion time ration of a task to memory allocated for that task, if-and-only-if the tasks completion speed varies
- T_{total} Is the total time taken to complete all tasks
- T_k Is the number of tasks in the application
- r_m is the memory available/used.

Similarly, we compute the cost associated with running a superstep by considering the time taken to run the application as well as the number of tasks within the application and the memory allocated to the application by using the following equation:

$$C_{total} = T_\alpha \times T_k \times C_{r_m} \tag{7.10}$$

Where:

- C_{total} is the cost associated with running k amount of tasks.
- C_{r_m} The cost associate with hiring resource of type m.

An important observation is to note is the cost associate with running applications.
In the scenario when the total running time of all tasks combined C_{total} is less than 60 minutes, then the cost of running that specific application will amount to the same cost since Amazon Web Services costs are calculated on an hourly basis. Thus, meaning that if a application runtime completes while $T_{total} < C_{r_m}$ then the total cost will still incur the full price for that hour. However, a good indication of the ration of the cos based on runtime may indicate that the resource type being used may not be necessary as well as open possibilities for sharing the resource that may not be used to

run multiple applications can be of benefit.
Furthermore, the model does not take into consideration of start-up times which for large cluster on AWS may take any time from seconds to a few minutes depending on cluster instance availability as well as the amount of instances being hired.

User interactions such as set-up and application installation were excluded from the results as it is difficult to obtain a specific amount of time that the user may take for miscellaneous activities such as application installation and uploading of datasets.

7.7 Cost and Performance Constraints

In this section, the results of the analysis between cost and performance for multiple variations of resources with varying quantities are analysed through the use of Pareto Frontier. Further, the results obtained we show a model of choosing the best resource type based on user requirements and preferences.

Firstly, in this experiment multiple resource types from AWS EC2 are analysed for their performance as well as cost.
Due to the relationship of cost and performance, in the scenario where both entities are improved without a trade-off is the most optimal solution considering that the choice offers the most improvement in both entities.
However, performance and cost in our results show that increasing performance by purchasing a higher resource amount directly affects cost.
Thus, the use of Pareto Frontier allows the elimination of ineffective possibilities while selecting only the options where a fair trade-off occurs. Firstly, finding the Pareto Frontier Algorithm 1 was implemented to perform the computations.

Algorithm 1: Pareto Frontier Sets

```
input: F {c, p} (pairs of all sets of points)
output: PF (c, p)
begin
    sort Fc \\ascending
    let c := Ccheapest ;
    add Fc to PF;
    while (p > c)
        if F(p) > Fc) then
            add F(c) to PF;
            let c := p;
            goto res;
        else
            break;
        end if;
    end while;
end begin;
```

Finding the Pareto Frontier, Algorithm 1 is utilised. Firstly the data obtained from testing a set of data across multiple AWS EC2 instance types to obtain performance(run time) as well as the cost associated with the computation. Tested instances included T, M, C and R. However, the results focused on r3 instances due to their optimization in terms of memory. These results can be replicated in terms of CPU. However graph drawing algorithms such as Gila used through Apache Giraph is memory intensive.

Model	CPUv	Memory(GB)	Storage SSD(GB)
r3-large	2	15.25	1x32
r3-xlarge	4	30.5	1x80
r3-2xlarge	8	61	1x160
r3-4xlarge	16	122	1x320
r3-8xlarge	32	244	2x320

Table 7.5: R3 Instance specifications.

In Figure 6.8, the data point in black represent the Pareto frontier computed through algorithm 1 in terms of cost and completion time. However, the data points in red, were excluded from the Pareto frontier for one of the following reasons:

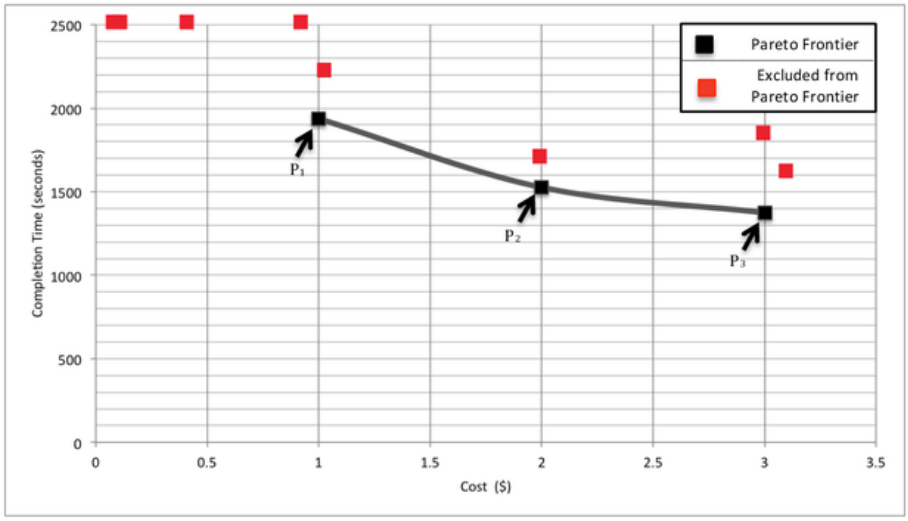


Figure 7.9: Pareto frontier for Gila graph drawing framework

- The red data points either exceeded a safe limit of completion time, or error incurred such as lack of memory and the inability to perform all the necessary computations. Noticeably, the data points P_1, P_2, P_3, represent the following instance r3-xlarge, r3.2xlarge and r3-4xlarge respectively.

Model	CPUv	Memory(GB)	Storage SSD(GB)
r3-xlarge	4	30.5	1x80
r3-2xlarge	8	61	1x160
r3-4xlarge	16	122	1x320

Table 7.6: R3 Instance specifications.

Instances r3-x8large, offered double the resource capacity, however the performance was worse-off when compared to the productivity of r3-4xlarge. Thus, despite the amount of resource for r3-8xlarge being doubled, the cost increased while performance shows worse-off when compared to r3-4xlarge.

7.7.1 User optimization selection

In Figure 6.9 Pareto Frontier includes only points P where P values correspond to instance types.
User based optimization requires *priori* of specific optimization targets, thus requiring user speciality on the subject as well as the difficulty to obtain an accurate objective may be challenging. However, the problem and optional choices can be simplified by providing ways to select a specific goal based on one parameter such as cost or performance. Furthermore, we suggest a simple model as shown in equation (6.11) that selects the most dominant value in terms of performance. Consequently, the instance type with the best performance is associated with a trade-off in terms of cost.
The set of choices is limited prior to user decision making through Pareto frontier. Only Pareto Frontier solutions are take into consideration throughout the following equations.
With a objective choice from the user, once the appropriate time or performance value has been chosen, the value point (cost, performance) that results from the equation is used to match with a corresponding instance type.
For example: If a user choice is to optimize the application in terms of performance, equation (6.11) is used to obtain the option from the Pareto frontier with the cheapest cost. Thus, in that scenario, P_1 point from Figure 6.9 would be selected which corresponds to r3-xlarge.

$$Time_{min} = \min_{\forall p \ \epsilon \ F} \{F(p)\}$$
$$s.t. \quad Cost = \max_{\forall c \ \epsilon \ F} \{F(c)\} \tag{7.11}$$

Where:

- Time is the run time of a particular application.
- F is the set of optimal Pareto Frontier choices.

Conversely from the previous optimization technique, the user may prefer a cost based constrain where the cost is the main priority. Similarly, equation (6.12) selects the minimum value in terms of cost. The minimum cost is similarly to (6.11) incurs a loss in terms of performance(running time).
Notably, the minimum cost based optimization in terms of Pareto frontier refers to the extreme end point on the Pareto Frontier graph.

$$Cost_{min} = \min_{\forall c \ \epsilon \ F} \{F(c)\}$$
$$s.t. \quad Time = \min_{\forall p \ \epsilon \ F} \{F(p)\} \tag{7.12}$$

Utilizing a lowest cost optimization technique may offer the most optimal solution in terms of cost. Using this approach provides the most cost effective solution, however it may not be suitable for other optimization needs that the user may have. Such scenario may include projects based on a limited fixed budget amount. Let budget refer to C_x referring to a user budget choice.

Through equation (6.12) the optimization focuses on the most cost effective solution. However a fixed budget which is common amongst research projects and grants, a user may prefer to select the best instance/resource option that is within the specified budget C_x. Assuming the priori includes the value of C_x, using equation (6.13), it is possible to obtain the best instance in terms of cost from the Pareto frontier choices F, such that the maximum value obtained is the largest value in set F smaller than C_x.

$$Cost_{max} = \max_{\forall c \ \epsilon \ F} \{F(c)\} \quad .s.t. \quad C_x < Cost_{min} \tag{7.13}$$

Noting the results from the Pareto Frontier solution, in reference to points $P1, P2 and P3$ if the user chosen budget $C_x < P1_c$ then the equation (6.13) would suggest P2 assuming that $C_x > P2_c$ is true, then P2 is the selected instance option referring to a r3-2xlarge instance. Conclusively, it would result on the budget being underutilized by a cost fraction loss of:

$$Cost_{loss} = C_x - Pk_c \tag{7.14}$$

Where Pk refers to P2 in the mentioned scenario. Despite of lack of optimality, this is case specific since AWS EC2 instances resources

may not be customizable, specifically in terms of memory. However, for storage and CPU intensive applications AWS offers extensive customizability as well as the ability to control I/O speeds. The benefits accumulate further when more machines are tested against the Pareto optimality. The use of the mentioned models extends to self-hosted distributed computing hardware as more optimal options can be found for the Pareto Frontier by improving customizability. Further, all the choices chosen through the use of the above equations leads to a Pareto efficient choice, however in terms of cost budget customizability the result is a efficient Pareto choice with a potential underutilization of cost in terms of $Cost_{loss}$. Thus, on cost based budgets, optimality is dependant on the available number of resources as well as the variety of resource which has a correlation with the number of optimal Pareto Frontier solutions.

7.8 Case study: Pareto optimization

In section 6.7, Pareto Frontier and cost/performance trade-off optimization models are introduced. In this section, experimental results of a graph drawing application are demonstrated. Firstly, this experiment focused on a graph data with the same amount of verticies and edges while graph degree remains constant with the graphs used in section 6.7 for consistency purposes. Further, following equation (6.12) for performance based optimization. Thus, this experiment seeks the most optimal solution in terms of performance disregarding cost.
From the use of equation (6.12), we obtain P_3 as the optimal instance type. where P_3 refers to instance r3.4xlarge as shown in table 6.7. Non-optimal instances are disregarded from table 6.7.

Model	Reference	Memory(GB)	Storage SSD(GB)
r3-xlarge	P_1	30.5	1x80
r3-2xlarge	P_2	61	1x160
r3-4xlarge	P_3	122	1x320

Table 7.7: Pareto Frontier instance choices

Further, using r3.4xlarge, cluster optimization through resource provisioning using equation (6.1) for optimizing the maximum amount

of workers in the cluster. Furthermore, we compare initial results obtained prior to any optimization as shown in Figure 6.10. Where:

- Initial results refers to the runtime of the same graph ran with no optimization technique applied on a r3.xlarge instance.
- Pareto Optimized shows runtime reduction after selecting an optimized instance type r3.4xlarge.
- Pareto Optimized + Worker Optimization refers to runtime obtained by using an optimal instance from the Pareto Frontier additionally to adjusting the number of workers within the cluster based on techniques introduced in section 6.1.

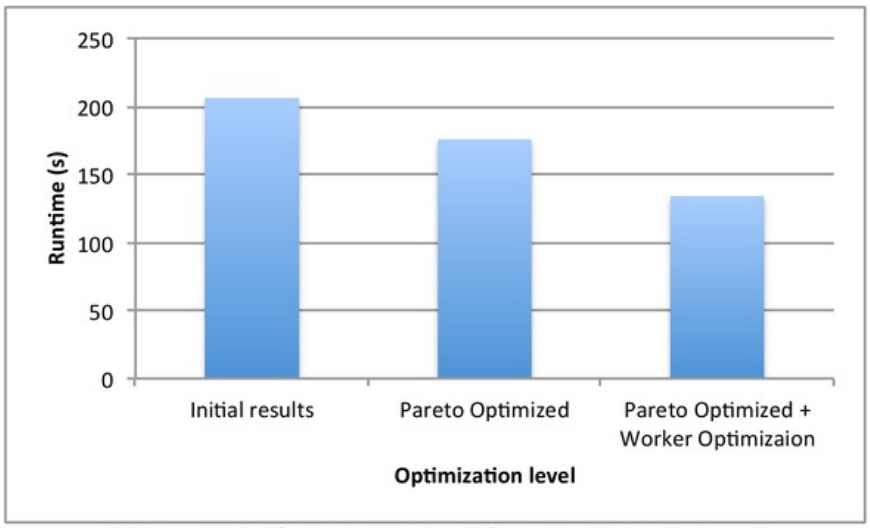


Figure 7.10: Optimization level for a 1 million edge graph

7.9 Case Study: Cluster Performance Analysis

Cluster performance behaviours are dependant on numerous factors that can affect the outcome and efficiency of running large scale application on cloud distributed resources such as Amazon Web Services. In this section, we analyse cluster behaviours based on performance optimization techniques developed throughout the previous sections. As a case study, graph drawing algorithms(Gila) were used on a distributed framework Apache Giraph. Firstly, experimental results based on running clusters with multiple amounts of instances are recorded in table 6.7. for numerous graph data of various sizes are tested.

Graph Name	Cluster Size	Edges	Runtime(sec)
add32	10	10,500	46
grund	10	17,427	51
com-Amazon	10	925,872	248
comDBLP	10	1,049,866	452
roadnetPA	10	1,541,514	328

Table 7.8: Real Graph data sets 10 r3.xlarge instances

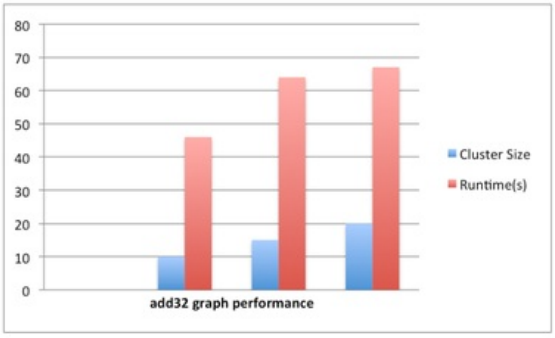
Graph Name	Cluster Size	Edges	Runtime(sec)
add32	15	10,500	64
grund	15	17,427	67
com-Amazon	15	925,872	201
comDBLP	15	1,049,866	369
roadnetPA	15	1,541,514	274

Table 7.9: Real Graph data sets 15 r3.xlarge instances

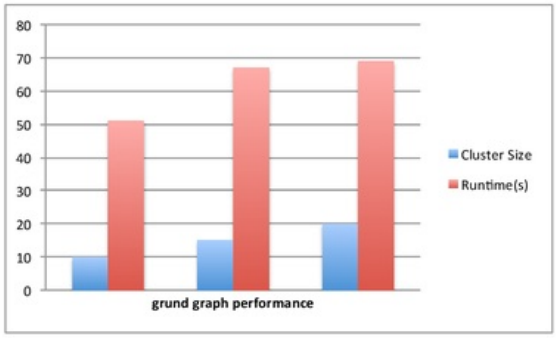
Graph Name	Cluster Size	Edges	Runtime(sec)
add32	20	10,500	67
grund	20	17,427	69
com-Amazon	20	925,872	173
comDBLP	20	1,049,866	360
roadnetPA	20	1,541,514	260

Table 7.10: Real Graph data sets 20 r3.xlarge instances

Table 6.8 shows the results of the same graphs tested previously, however the cluster configuration is changed to accommodate more instances. In all the the results in tables 6.7 to 6.9, each instance is allocated a single worker, thus maximizing the resource and reducing the number of overall graph partitions. Consequently reducing the number of verticies migration during each superstep and increasing locality of edges within each compute() unit. In order to analyse the effect of cluster, from table 6.7 to 6.9 we analyse scalability based on graph data size in terms of vertices and edges.

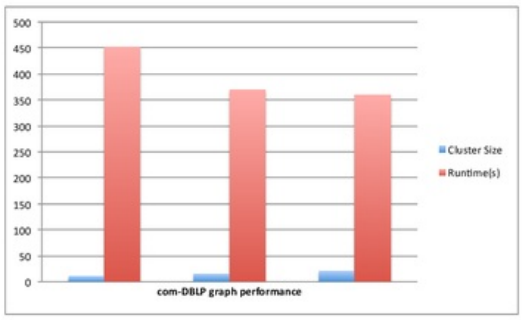


(a) Performance of add32 with 10,500 edges

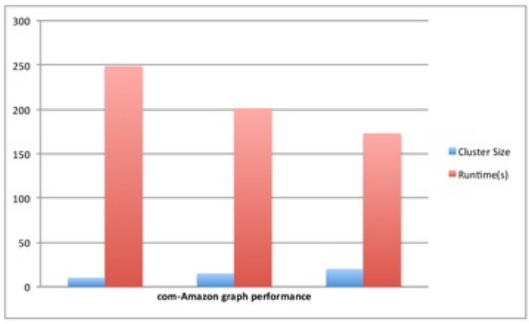


(b) Performance of grund with 17,427 edges

Diagrams (a) and (b) both show decrease in performance as cluster size increases where performance in runtime is expected to improve. However, despite the slight reduction in performance is assignable to the fact that cluster size ranges from 10 to 20. Each machine requires its own JVM as well as edge locality and migrations between each instance/worker affects runtime significantly impacting graphs with small amount of edges, that is graphs under 50,000 edges.

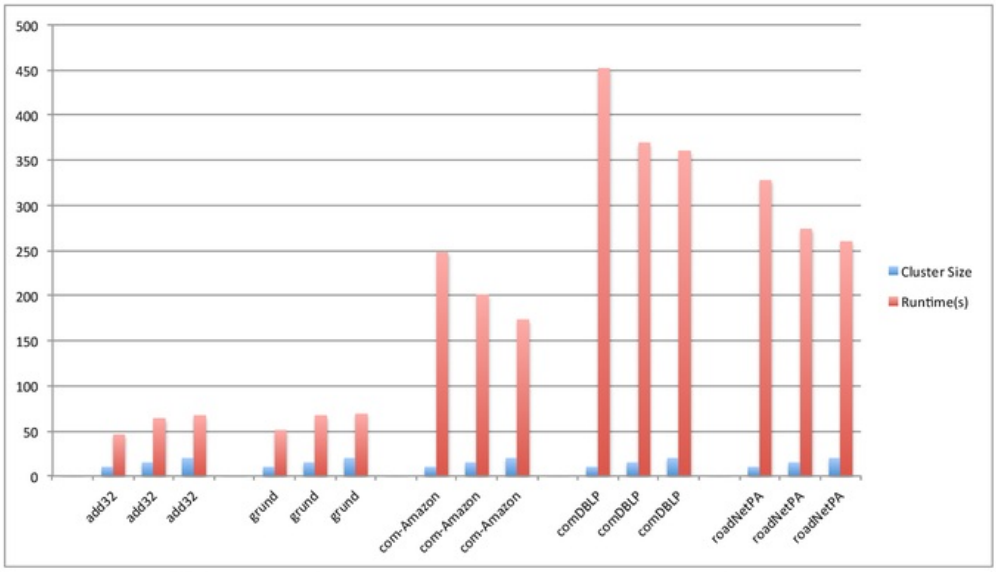


(c) Performance of com-DBLP with 1,049,866 edges



(d) Performance of com-Amazon with 925,872 edges

Conversely, the number of edges directly correlates with cluster performance. The number of instances in a cluster reduces the overall runtime as illustrated in diagram (c) and (d) where the number of edges are approximately 1 million. In terms of scalability, the number of machines from 10 to 20 shows scalable performance gain as the number of edges increase as shown in diagram (e).



(e) Cluster performance: graphs from 10 thousand to 1 million edges

The importance of cluster scalability is especially critical with larger graphs. Conclusively, it may prove difficult selecting between cluster sizes and configurations. However, it is crucial user choices are curated on the basis of performance.
Computing performance level of a cluster composition can determine the scalability of resources in terms of execution speed. Performance level for a cluster composition can be formulated by the following equation:

$$P_i = \frac{\frac{V_n}{E_n}}{I \times T} \tag{7.15}$$

Where:

- V_n is the amount of vertices in the graph.
- E_n is the amount of edges in the graph.
- I is the number of machines in the cluster.
- T is the time taken by the application to complete.

Utilizing equation (6.15), a diagram illustration shows the change in performance level increases as the number of workers is increased. The benefits of calculating performance level of a cluster, allows the user to observe how a certain data set behaves in multiple cluster compositions. Consequently, if multiple graph data sets of similar attributes are to be processed, selecting a cluster size based on performance level may improve overall performance. In figure 6.10, performance index improves as the amount of in-

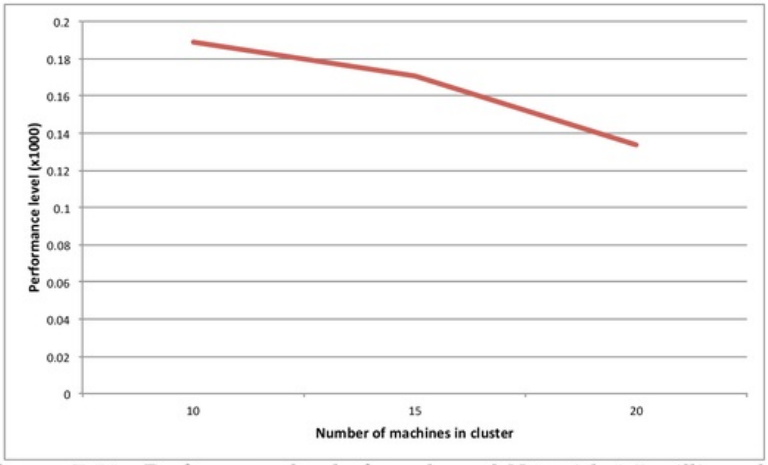


Figure 7.11: Performance level of graph road-Net with 1.5 million edges

stances improves from $10 > 15 > 20$. Performance level improvement is considered as performance level in Figure 6.10 approaches 0. Conclusively, performance level is an indicator of cluster performance in terms of runtime without factoring cost increases as hiring more machines increases overall running cost. The trade-off between cost and performance has been analysed in section 6.7.

Chapter 8

Discussion

In the previous chapters we have presented the research conducted in this project. Here, we discuss the results obtained based on the introduced methodologies developed throughout this project and its implications in various areas of distributed cloud resource provisioning. We begin by discussing the importance of curated resource provisioning and scenarios where the methodologies developed in this project may assist in selective decision making. Additionally, we discuss how Pareto optimality and the introduced models assist by filtering optimal solutions reducing the number of solutions that the user is presented with.

8.1 Instance Worker Optimization

In section 6.1 we discussed the correlation between the number of workers per instance within a cluster. Firstly, the results showed that an increase in the amount of workers has an impact on execution speeds. The results showed that using one worker per instance results in better execution speeds regardless of algorithm.
Intuitively, increasing the number of workers reduces the workload for each worker in terms of computation, this is due to the fact of graph partitioning being split across each worker for a balanced distributed system. As the number of workers increase, the percentage of graph locality decreases. With a decrease in edge locality, the number of migrations after each superstep(iteration) increases to produce a balanced partitioning of the graph. Furthermore, the reduction is edge locality between workers was not necessarily the underlying reason for such runtime increase.

The main factor of worker count correlating to increased runtime is due to resource competition between each worker as they compete for the same resources within an instance. Furthermore, each worker is launched within its own container as a virtual machine where auxiliary system support applications are launched further reducing the amount of resources available for computational requirements. Resource competition decreases as resource availability increases on each instance to the point where increasing the number of workers creates benefits including runtime reduction.

8.2 Consumption Patterns

Throughout this project, whilst testing and executing graph drawing applications as well as graph processing algorithms, one of the major issues highlighted was determining the amount of memory required for successfully completing a graph drawing. In section 6.2, we analyse the lower bound of resource consumption footprint of graphs in terms of verticies and edges. Through the data collected, we formulated a method of determining the lower bound of memory required to successfully run a graph drawing job. The importance of acquiring a lower bound even at approximate levels, reduces the chances of resource under allocation.

Special factors need to be taken into consideration, including the type of graph attributes such as graph degree and the ratio of verticies to edges.

The benefits of eliciting requirements in terms of resources benefits future computations, as optimization techniques are applied. Scenarios that would benefit from consumption pattern and graph footprint analysis would include data that is constantly changing with graph degree and other graph attributes remaining constant. Further applications that benefit include scientific workflow ensembles, where consumption patterns can be discovered in terms of application tasks due to their data attributes remaining immutable.

8.3 Provisioning Models

As resource patterns are encapsulated, allocating resources based on availability is a critical function. During our research, we intro-

duced methodologies of curating resource allocation based on cluster circumstances and resource availability. Benefits of such analysis provides a certain sequential calculation of optimizing cluster resources based on factors that dynamically change in distributed cloud computing. Such factors include:

- Instance types
- Cluster composition, homogeneous vs heterogeneous resource compositions
- Instance availability (Specific to Amazon where regions can affect cost and availability).

8.4 Pareto Optimality

In regards to resource allocation in cloud environments, several conflicting incomparable entities rise. Performance and cost in most cases require a trade-off. Enhancing performance consequently requires a more expensive cluster composition. Conversely, cheaper objectives result in decrease in performance. The use of Pareto optimality and Pareto Frontier analysis provides clear insight into the underlying truth of options available to the user.

Through section 6.5, We introduce several ways of optimising the selection of options based on user objectives. Since user objectives are difficult to obtain, and, possible user requirements are dynamic in terms of customizability, it is critical that the introduced methodologies target the problem of optimal choices based on user requirements. Through Pareto Frontier analysis, we demonstrated that it is possible to obtain a set of optimal solutions that lead to an increase in performance or cost, depending on user requirements.

Common scenarios where the need to select the most optimal option amongst multiple choices with a limited and specific budget can include research projects where grants are allocated. In such cases, where research grants or business budgets are set and unchangeable, selecting the most optimal resources in terms of performance without exceeding the budget is critical to the projects success.
Our results through case studies demonstrated that through the use of Pareto Frontier filtering, as well as the introduced models

for maximizing resource allocation, performance improvements up to 30% can be achieved, which in a long term project with hundreds to thousands of tasks, will consequently lead to reduction of the budget required or enhancement of execution speeds for applications in distributed cloud environments.

Chapter 9

Conclusions

The goal of this project was to assess resource provisioning patterns for cloud based distributed computing though graph drawing applications and determining models for optimal resource allocation.
Through our research, we discovered the lack of precise models for allocating resources based on data footprints. To satisfy this, we conducted experiments to elicit graph resource usage footprints. Through the use of the models based on gathered data, we were able to successfully complete multiple applications without resource under allocation issues.
Moreover, additional models were introduced that provide efficient resource allocation based on multiple user objectives.
In this chapter, we conclude with the research conducted throughout this project by examining the objectives accomplished.

A literature review was conducted based on recent research in distributed cloud computing in order to understand the fundamental concepts required for conducting this project. Further background research was gathered in the field of graph processing, more specifically with a focus on distributed graph drawing.
In the following chapters, experimental and risk assessment strategies were conduced for maximizing project success and handling potential risks that may have risen during the experiments. Concluding risk assessment, experiments were designed in incremental stages for improving resource provisioning and designing scheduling solutions.
From the initial experiments, we concluded results and introduced models for resource allocation problems through case studies in graph drawing applications. It was discovered that the need for

precise allocation models are required for preventing issues in regards to over or lack of resource consumptions. Through visualization of resource consumption patterns of graph processing algorithms, we developed several optimization models and techniques for assisting users of cloud resources to make optimal decisions in terms of resource selection for maximal objective reach. Through optimization techniques and introduced models, we demonstrated the ability to increase performance in terms of cost and performance.

Chapter 10

Future Work

Based on the data collected in graph resource consumption patterns, future work may include the expansion of experiments towards scientific workflow applications where data patterns can be used for curated resource allocation. Expanding the introduced models to apply to a wider scope of distributed applications would also provide margin for future research.

10.1 Design and Implementation

Further enhancement and implementation of a framework that uses numerous data sampling techniques for eliciting application behaviours based on data input, consequently, reducing the need for user priori, increasing the accuracy of application resource consumption predictability.

10.2 Resource Consumption Sampling

Implementation and design of resource consumption sampling automation to reduce user priori to determine data properties and attributes. Alternatively, data sampling can be used to elicit resource consumption based on input data rather than pre-computed approximations. Performing sampling techniques, reduces possible user error during calculation and approximation stages.

Chapter 11

Abbreviations

AWS	Amazon Web Services
EC2	Elastic Compute
HDFS	Hadoop Distributed File System
BSP	Bulk Synchronous Parallel
JVM	Java Virtual Machine

Chapter 12

Definitions

Instance	A physical machine which consists of multiple resources such as Memory and CPU.
Worker	A processing unit capable of performing computations on behalf of a Application/Algorithm.
EC2 Region	A Chosen geographical region where the physical instances are hosted
Cluster	A collection of resources from multiple machines hired on demand

Appendix A

Appendix A

A.1 Instance Pricing - Sydney Region

In the table below, the all the prices have been included. The prices listed have been used for experimental calculations and modelling. For experiment replication and further research, the prices for EC2 instances may change frequently.

Instance Type	OS	Cost per Hour
t2 nano	Linux	$0.01
t2 micro	Linux	$0.02
t2 small	Linux	$0.04
t2 medium	Linux	$0.08
t2 large	Linux	$0.16
m4 large	Linux	$0.168
m4 xlarge	Linux	$0.336
m4 2xlarge	Linux	$0.673
m4 4xlarge	Linux	$1.345
m4 10xlarge	Linux	$3.363
m4 16xlarge	Linux	$5.381
m3 medium	Linux	$0.093
m3 large	Linux	$0.186
m3 xlarge	Linux	$0.372
m3 2xlarge	Linux	$0.745
c4 large	Linux	$0.137
c4 xlarge	Linux	$0.275
c4 2xlarge	Linux	$0.549
c4 4xlarge	Linux	$1.097
c4 8xlarge	Linux	$2.195
c3 large	Linux	$0.132
c3 xlarge	Linux	$0.265
c3 2xlarge	Linux	$0.529
c3 4xlarge	Linux	$1.058
c3 8xlarge	Linux	$2.117
g2 2xlarge	Linux	$0.898
g2 8xlarge	Linux	$3.592
x1 16xlarge	Linux	$9.671
x1 32xlarge	Linux	$19.341
r3 large	Linux	$0.2
r3 xlarge	Linux	$0.399
r3 2xlarge	Linux	$0.798
r3 4xlarge	Linux	$1.596
r3 8xlarge	Linux	$3.192
i2 xlarge	Linux	$1.018
i2 2xlarge	Linux	$2.035
i2 4xlarge	Linux	$4.07
i2 8xlarge	Linux	$8.14
d2 xlarge	Linux	$0.87
d2 2xlarge	Linux	$1.74
d2 4xlarge	Linux	$3.48
d2 8xlarge	Linux	$6.96

Table A.1: Amazon Instance prices - Sydney Region

A.2 Instance Resource Specifications

Throughout the experiments, various instance types were used. Each instance type has a specified amount of resources. However the resources available for each type are subject to change, thus, in the table A.2 instance properties are recorded.

Model	CPUv	Memory(GB)	Storage SSD(GB)
r3-large	2	15.25	1x32
r3-xlarge	4	30.5	1x80
r3-2xlarge	8	61	1x160
r3-4xlarge	16	122	1x320
r3-8xlarge	32	244	2x320

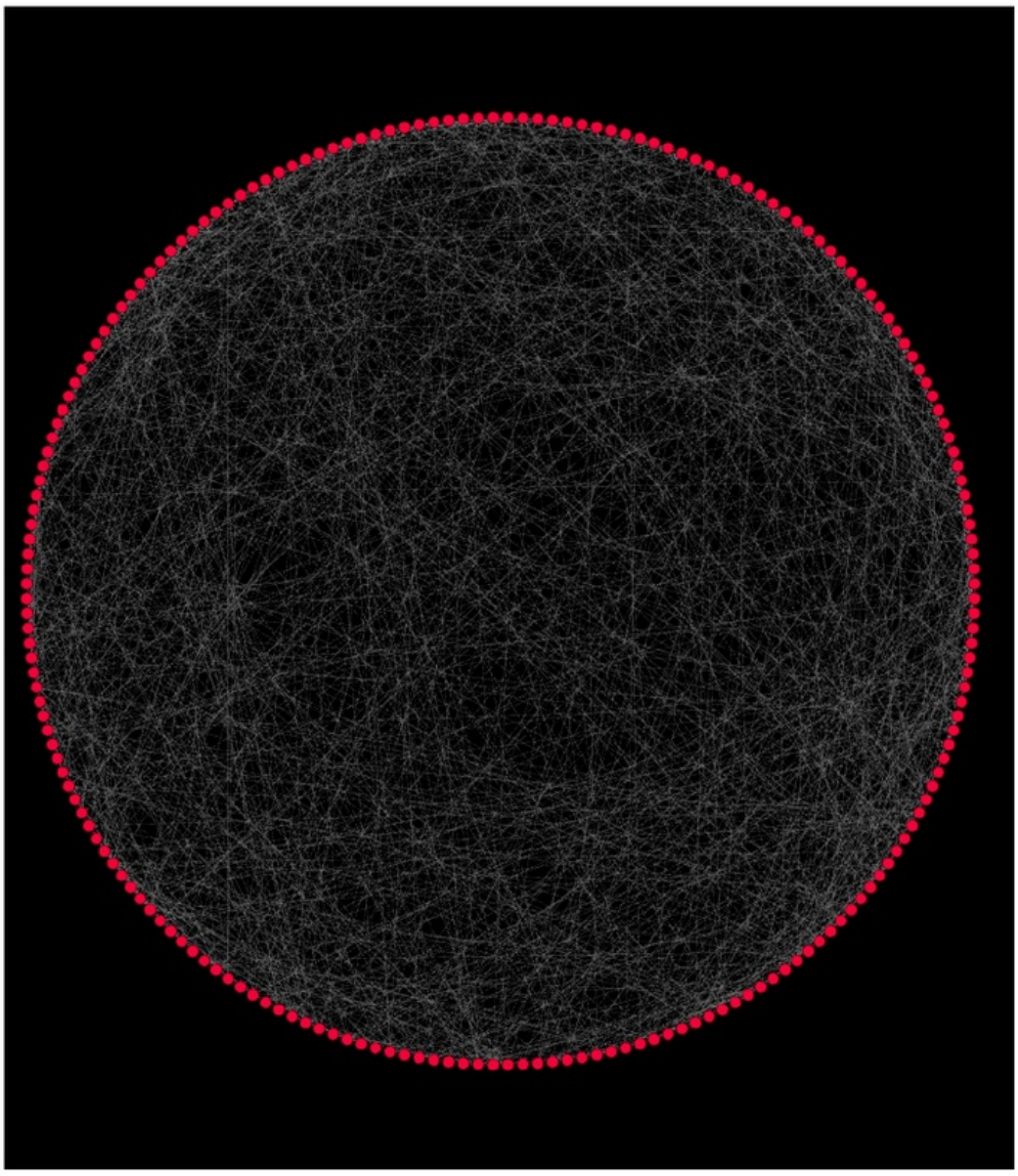
Table A.2: R3 Instance specifications.

Appendix B

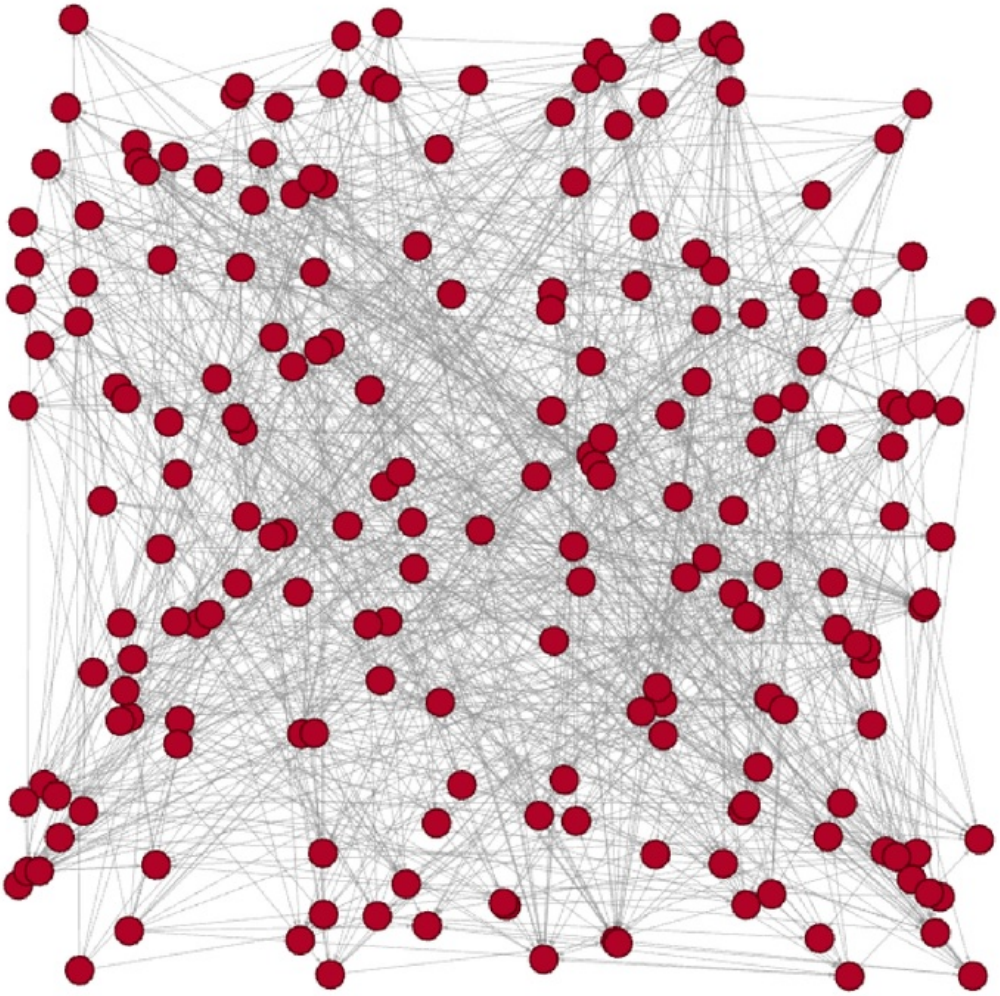
Appendix B

B.1 Graph Layout Visualization

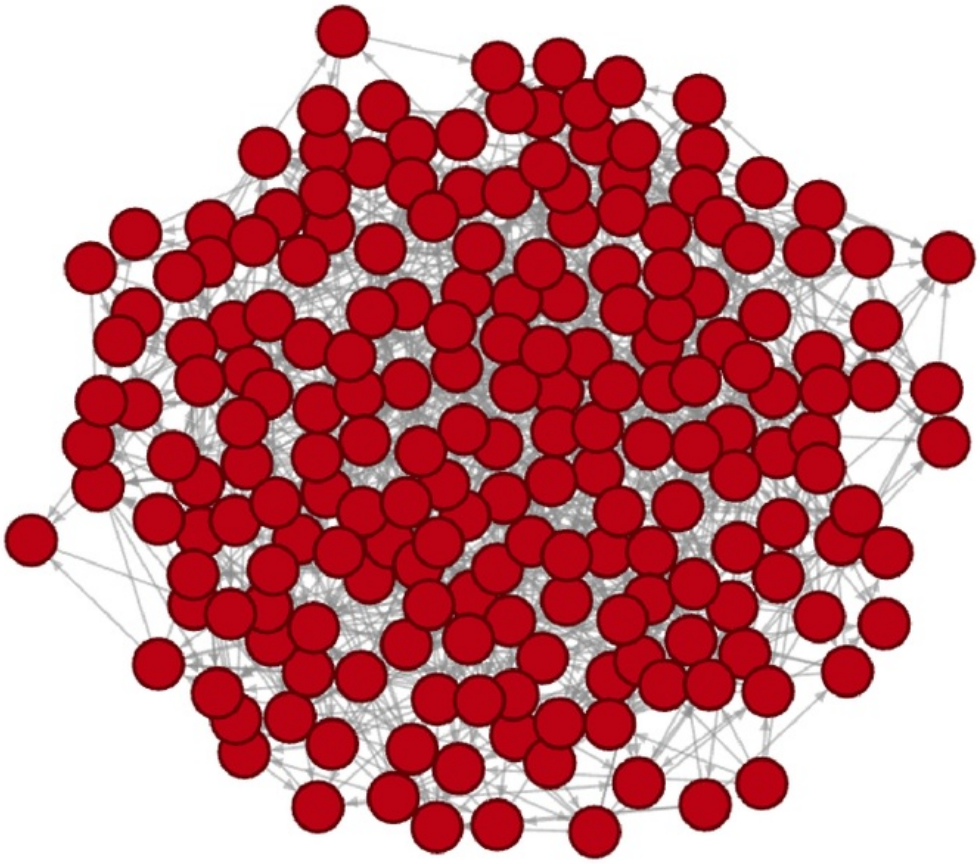
In this section, we produce multiple layout algorithms that we analysed during the research phase of this project. The layouts displayed in this section represent the same graph with various graph visualization layouts.



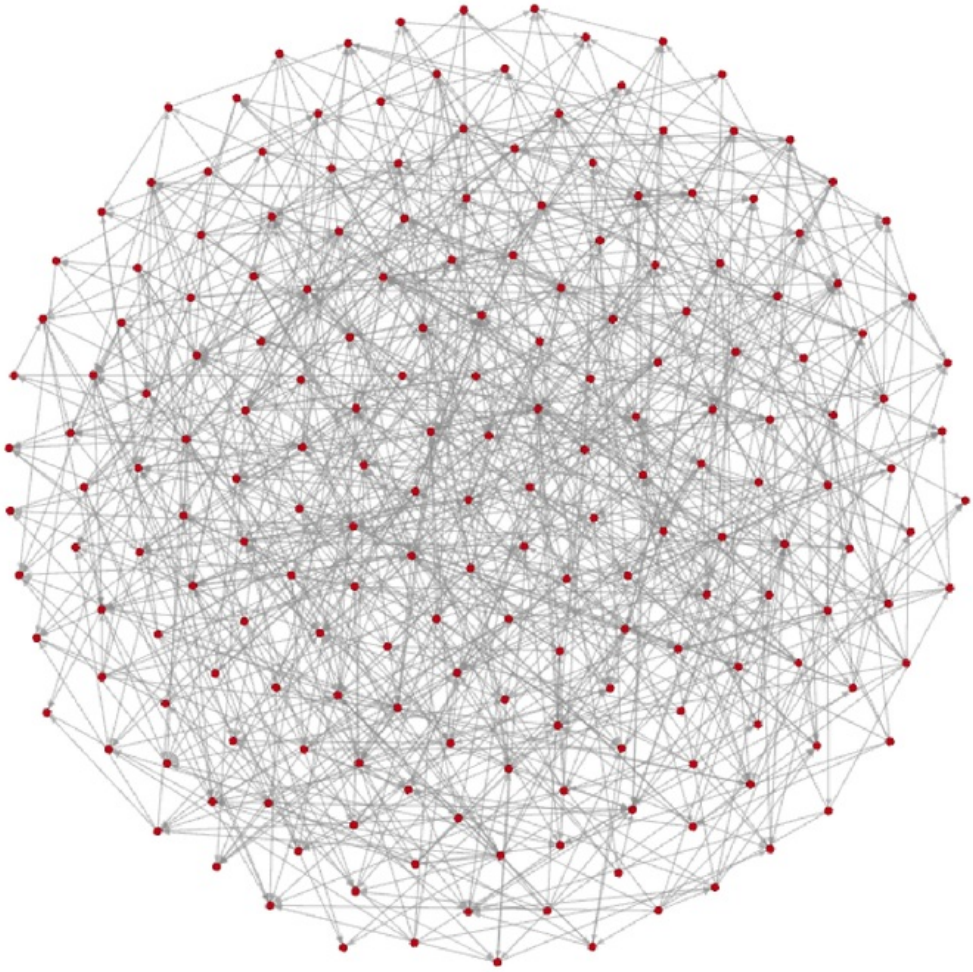
(a) Graph layout: Circular



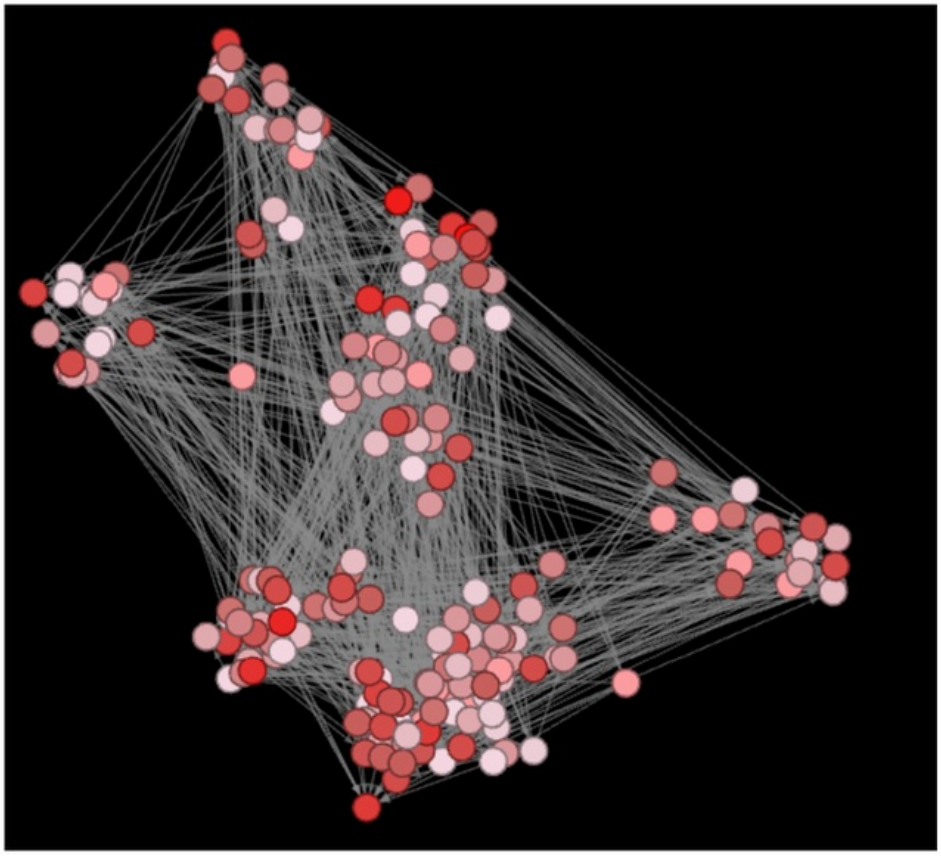
(b) Graph layout: Contraction



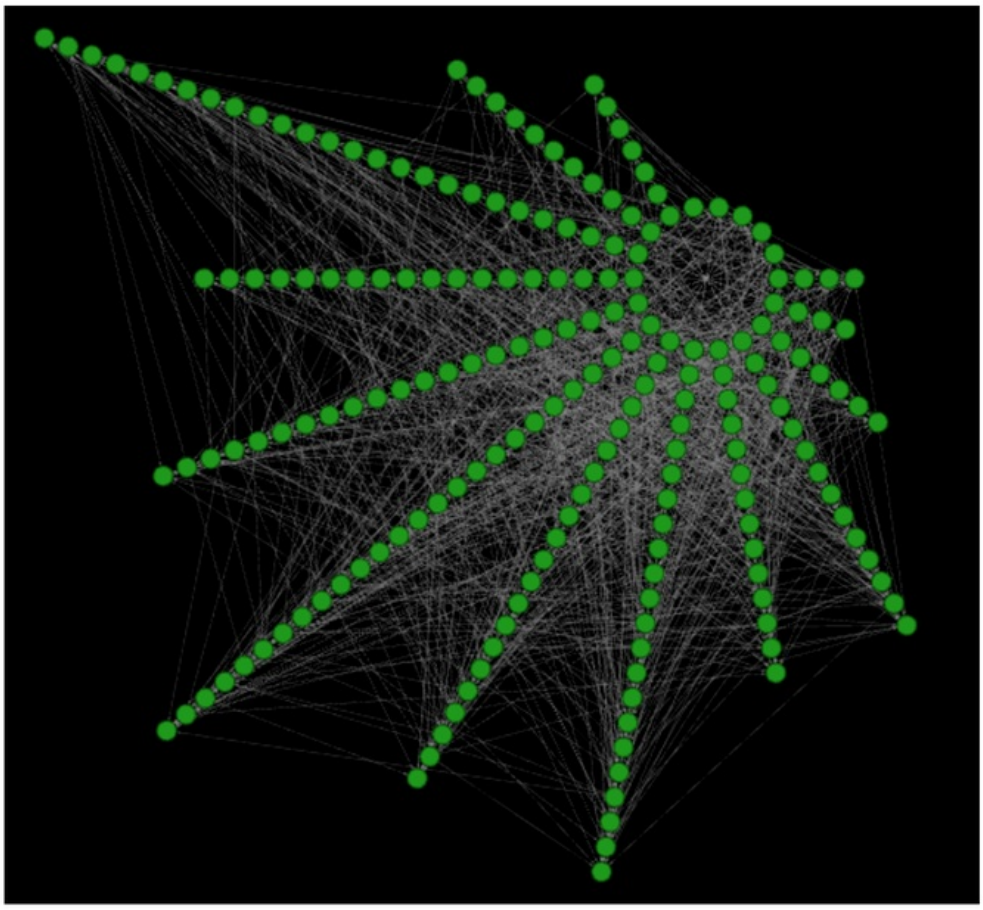
(c) Graph layout: Force Atlas



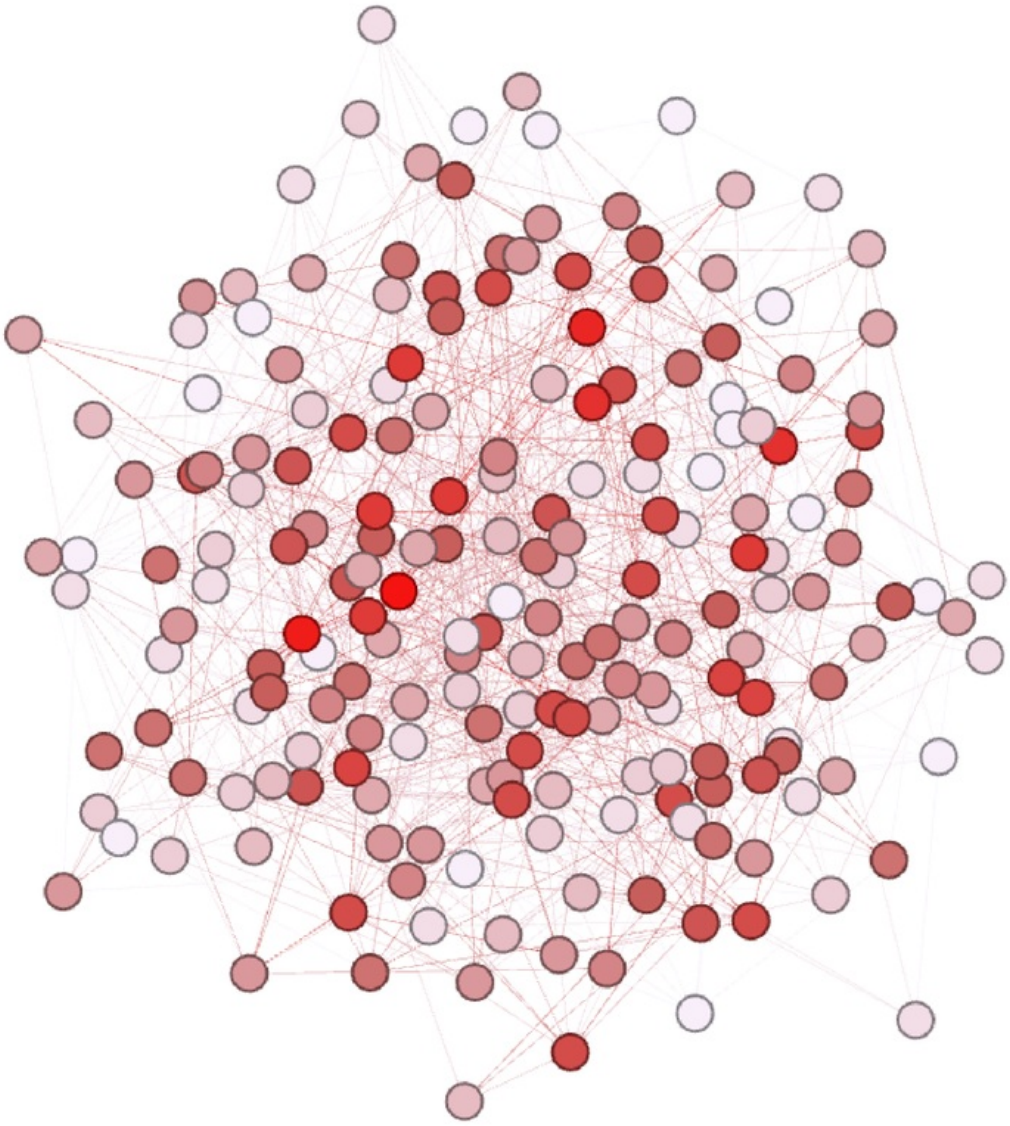
(d) Graph layout: Fruchterman Reingold



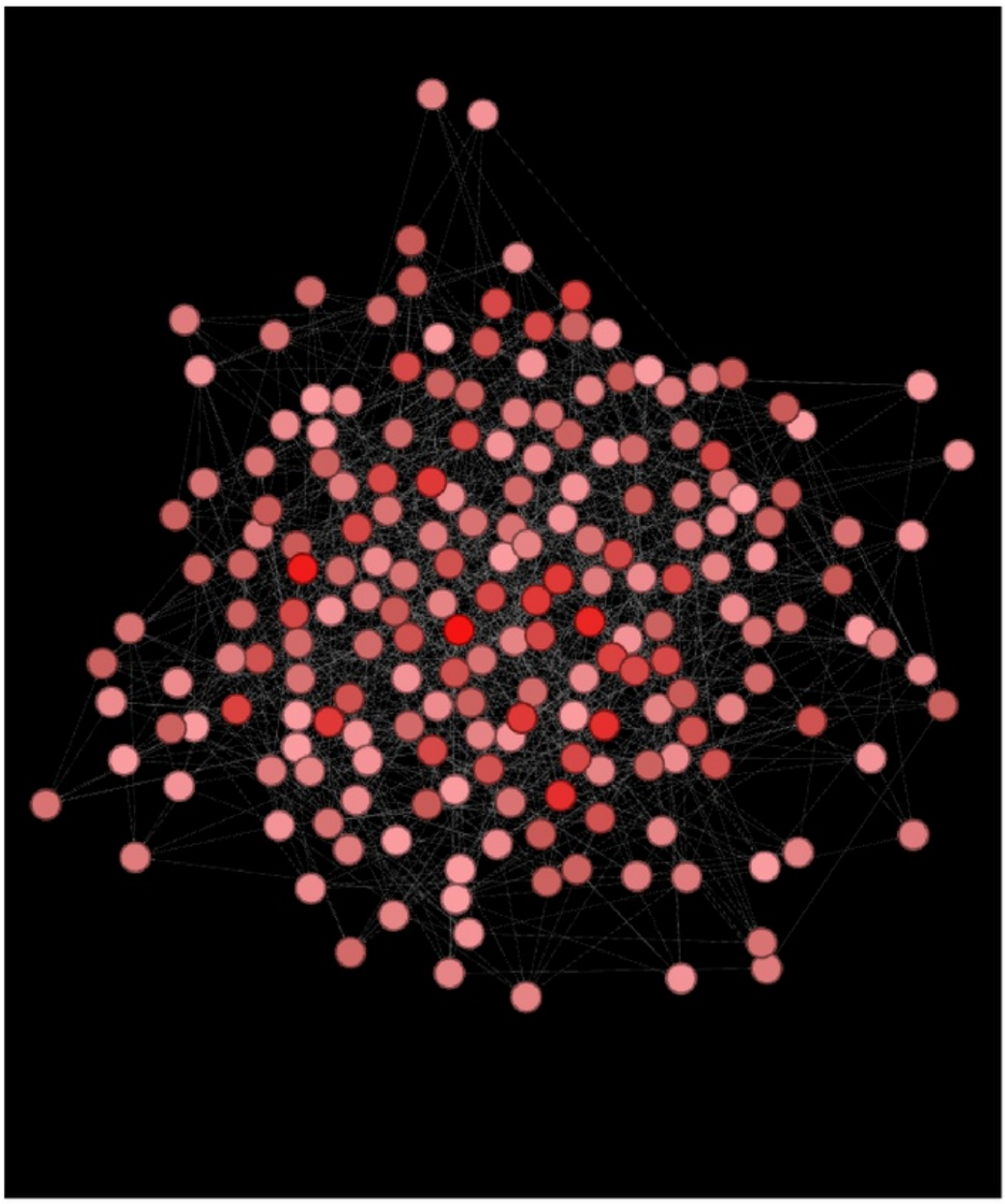
(e) Graph layout: OpenOrd



(f) Graph layout: Radial Axis



(g) Graph layout: Yifan Hu

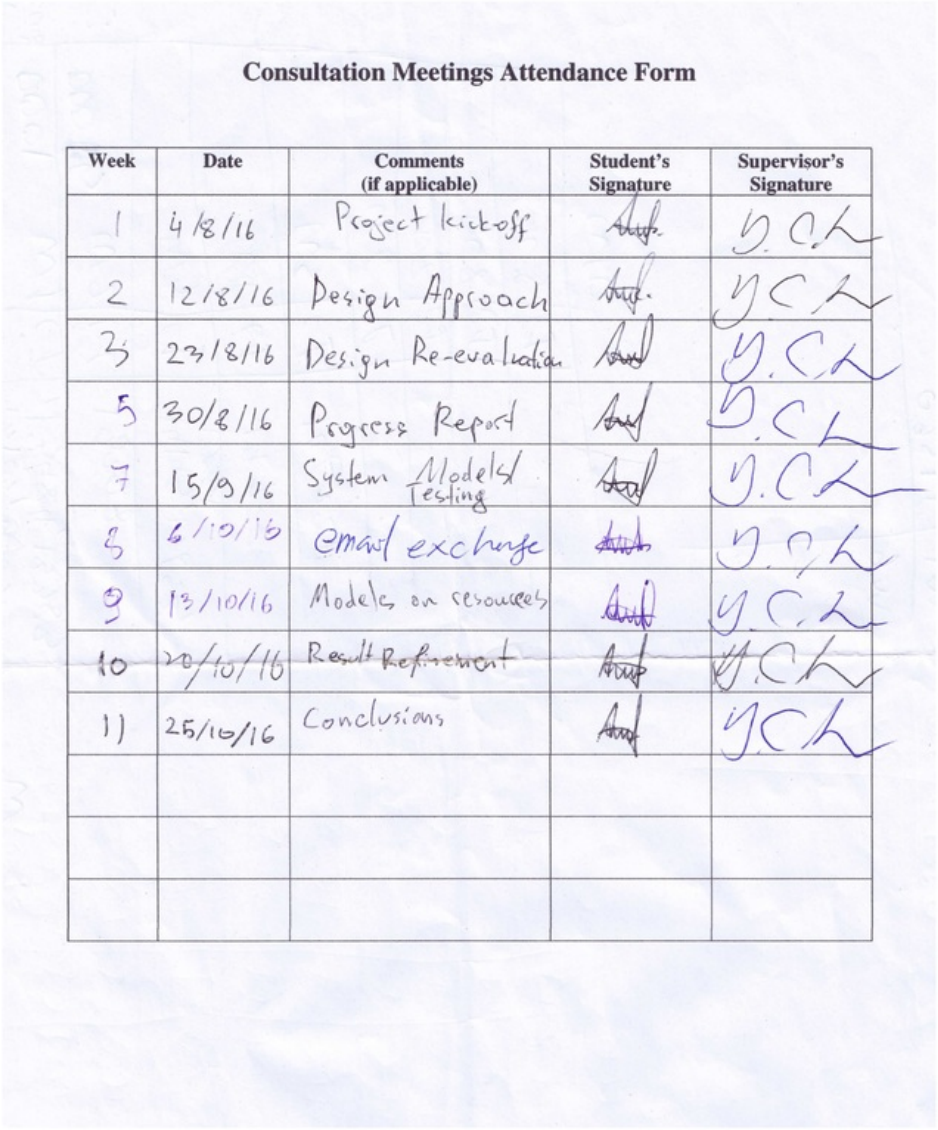


(h) Graph layout: Yifan Hu Proportional

Appendix C

Appendix C

C.1 Consultation Meetings



Consultation Meetings Attendance Form

Week	Date	Comments (if applicable)	Student's Signature	Supervisor's Signature
1	4/8/16	Project kick-off		
2	12/8/16	Design Approach		
3	23/8/16	Design Re-evaluation		
5	30/8/16	Progress Report		
7	15/9/16	System Modeld/ Testing		
8	6/10/16	email exchange		
9	13/10/16	Models on resources		
10	20/10/16	Result Refinement		
11	25/10/16	Conclusions		

Figure C.1: Consultation meetings

Bibliography

[1] I. Bermudez, S. Traverso, M. Munafò, and M. Mellia, "A Distributed Architecture for the Monitoring of Clouds and CDNs: Applications to Amazon AWS ," *IEEE Transactions on Network and Service Management, Vol. 11, No. 4, 516-529*, 2014.

[2] M.-H. Chiang, M.-C. Chiang, and C.-S. Yang, "A hyper-Heuristic Scheduling Algorithm for Cloud," *Cloud Computing, IEEE Transactions on 2.2, 236-50*, 2014.

[3] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Association for Computing Machinery. Communications of the ACM 51.1, 107-13*, 2008.

[4] C. Eddy, R.-M. Luis, D. Frédéric, and M. Adrian, "Auto-Scaling, Load Balancing and Monitoring in Commercial and Open-Source Clouds ," *INRIA, 27*, 2012.

[5] A. Fox, M. Armbrust, R. Griffith, R. Katz, and A. Konwinski, "A view of Cloud Computing," *Communications of the ACM 53.4, 50.*, 2010.

[6] P. Gajdoš, T. Ježowicz, V. Uher, and P. Dohnálek, "Add to e-Shelf A parallel Fruchterman–Reingold algorithm optimized for fast visualization of large graphs and swarms of data ," *Swarm and Evolutionary Computation, Vol. 26, 56-63*, 2016.

[7] J. Gideon, E. Deelman, B. Berman, and P. Maechling, "An Evaluation of the Cost and Performance of Scientific Workflows on Amazon EC2." *Journal of Grid Computing 10.1, 5-21*, 2012.

[8] J. Gideon, E. Deelman, K. Vahi, G. Mehta, B. Berriman, and P. Maechling, "Scientific Workflow Applications on Amazon EC2." *n. pag.*, 2010.

[9] J. Gonzalez, Y. Low, A. Kyrola, C. Guestrin, D. Bickson, J. M. Hellerstein, and C. Guestrin, "Distributed GraphLab: A Framework for Machine Learning in the Cloud ," *Proceedings of the VLDB Endowment (PVLDB), Vol. 5, No. 8, 716-727*, 2012.

[10] S. Goraya and K. Vikas, "Auto-Scaling, Load Balancing and Monitoring in Commercial and Open-Source Clouds ," *International Journal of Computer Applications, Vol. 121, No. 12, 133*, 2015.

[11] Q. henhao, C. R. N, and B. Rajkumar, "A reliable and cost-efficient auto-scaling system for web applications using heterogeneous spot instances ," *Journal of Network and Computer Applications, Vol. 65, 167-180*, 2016.

[12] I. Kacem, S. Hammadi, and P. Borne, "Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic ," *Mathematics and Computers in Simulation, Vol. 60, No. 3, 245–276*, 2002.

[13] S. G. Kobourov, "Spring Embedders and Force Directed Graph Drawing Algorithms. ," *arXiv:1201.3011, vol. 1 (PVLDB), Vol. 5, No. 8, 1-23*, 2012.

[14] P. Kokkinos, A. Varvarigou, A. Kretsis, P. Soumplis, and A. Varvarigos, "SuMo: Analysis and Optimization of Amazon EC2 Instances ," *Grid Computing, Vol. 33, No. 2, 255–274*, 2015.

[15] H. Li, B. Zhang, T. Gunarathne, S. hee Bae, J. Qiu, G. Fox, S. Hariri, and K. Keahey, "Twister: A Runtime for Iterative MapReduce. ," *N.p.: Acm. 810-18*, 2010.

[16] W. Ma, H. Zhang, Q. Li, and B. Xia, "Analysis of information management and scheduling technology in Hadoop ," *Journal of Digital Information Management, Vol. 12, No. 2, 133*, 2014.

[17] W. K. Mashwani, "Hybrid Multiobjective Evolutionary Algorithms: A Survey of the State-of-the-art ," *IJCSI International Journal of Computer Science Issues, Vol. 8, No. 3, 1694-0814*, 2011.

[18] C. A. Mattson and A. Messac, "Concept Selection Using s-Pareto Frontiers ," *AIAA Journal, Vol. 41, No. 6, 1190-1198*, 2003.

[19] G. Mone, "Apache Hadoop's continuous evolution," *Communications of the ACM, 56(1), 22*, 2013.

[20] G. Prasad, P. Swathi, G. Prasad, N. H, and P. Swathi, "Performance Analysis of Schedulers to Handle Multi Jobs in Hadoop Cluster ," *International Journal of Modern Education and Computer Science, Vol. 7, No. 12, 51-56*, 2015.

[21] S. Sakr, "Learn about Apache Giraph, GraphLab, and other open source systems for processing graph-structured big data," *Processing large-scale graph data: A guide to current technology. 5.*, 2013.

[22] R. Shawi, A. Fayoumi, R. Nouri, A. Barnawi, and S. Sakr, "Large Scale Graph Processing Systems: Survey and an Experimental Evaluation." *Cluster Computing 18.3, 1189-213*, 2015.

[23] R. Shawi, A. Fayoumi, R. Nouri, S.-M.-R. Beheshti, A. Barnawi, and S. Sakr, "Large Scale Graph Processing Systems: Survey and an Experimental Evaluation ," *Cluster Computing 18.3, 1189-213*, 2015.

[24] K. Shvachko, "HDFS scalability: the limits to growth," *VOL. 35*, 2010.

[25] H. Stefan, "Information-rich visualisation of dense geographical networks ," *Routledge , Vol. 9, No.1 68-75*, 2013.

[26] G. Sukhmani and K. Vikas, "Amazon Web Services Launches New Features for Amazon EC2 Enabling Monitoring, Auto Scaling and Elastic Load Balancing ," *Business Wire, n.p*, 2009.

[27] M. Tichy and H. Giese, "Implementing graph transformations in the bulk synchronous parallel model ," *Fundamental Approaches to Software Engineering, Vol. 17, 325-339*, 2014.

[28] T.M.J.Fruchterman and E. Reingold, "Graph drawing by force directed placement. ," *Software, Practice and Experience, Vol. 21*, 1991.

[29] L. G. Valiant, "A bridging model for parallel computation ," *Communications of the ACM, Vol. 33, No. 8, 103-111*, 1990.

[30] B. Yingyi, B. Howe, M. Balazinska, and M. Ernst, "The HaLoop Approach to Large- Scale Iterative Data Analysis." *The VLDB Journal 21.2, 169-90.*, 2012.