Efficient Parallel Data Processing in the Cloud

Ashley Ingram

BSc Computer Science

2013/2014

Summary

Fill in the summary here.

Acknowledgements

Most of all I’d like to thank my project supervisor . . .

Contents

Chapter 1

Introduction

1.1 Project Aim

Large scale data processing is an emerging trend in Computing, with benefits to both academia and industry. The aim of this project is to investigate the challenges of large scale parallel data processing, discuss the benefits of Cloud Computing and investigate how the efficiency of large scale data processing may be improved.

This project will aim to provide a reasoned and objective evaluation of the efficiency of data processing techniques, whilst comparing current state of the art tools and technologies with newer research in the field.

Specifically, the data processing tools Hadoop and Nephele will be compared, allowing comparison of the individual tools, and a wider discussion of the pros and cons of the MapReduce and PACT programming paradigms.

1.1.1 Research Questions

The project aims to answer the following questions:

- Does PACT overcome the inherent disadvantages of the MapReduce paradigm?
- How well does Nephele perform MapReduce tasks?
- How do Hadoop and Nephele perform in a highly elastic Cloud Computing environment?

1.2 Objectives

The objectives of the project are:

- Investigate Cloud Computing, Big Data and other relevant background trends
- Design a series of experiments to determine the efficiency of data processing technologies
- Implement the experiments for both Hadoop and Nephele
- Evaluate the experiment results to draw meaningful conclusions to the research questions

1.3 Methodology

The project will be broken into three main phases. The order to the phases provides a structured approach to the project, and completion of each phase ensures the project is completed in a methodical manner. The **Background Research** section will provide a comprehensive literature review to provide context to the project. This will be followed by a phase of **Experimentation**, which will aim to test the necessary factors of Hadoop and Nephele to answer the research questions. Finally, the **Evaluation** phase will evaluate the results of the experiments and the project as a whole.

The phases of the project will be performed according to Agile principles. Work will be carried out in weekly sprints, with clearly defined targets for each sprint. Supervisor meetings will be used as an opportunity to reflect on the achievements and shortcomings of the previous week, and to set targets for the upcoming sprint.

1.3.1 Background Research

The background research will aim to provide context for the project by surveying the current research landscape. It will summarise the fundamental concepts underlying the project and give a strong theoretical foundation for the rest of the project.

The first section of the background research will focus on Cloud Computing. It will explore the principles of the cloud, the service and deployment models and the technologies which make the cloud possible. It will also look at the unique advantages that the cloud provides in data processing scenarios.

The second segment of the background research will look in to Big Data and what challenges it introduces. An overview will be given of tools and techniques which have been developed to process Big Data, including Hadoop and Nephele.

1.3.2 Experimentation

Experiments will be designed to test the efficiency of data processing tools in relevant situations. This will require identifying relevant areas of interest, and deriving scenarios which will test the efficiency of Hadoop and Nephele. The experiments should be designed to take into account the various different problems that data processing tools are required to solve and to provide findings for the research questions.

As part of the experiment design, a hypothesis will be produced.

The experiments will be implemented and executed, in order to provide results which can be used to answer the research questions.

1.3.3 Evaluation

The project can be evaluated on various factors.

A technical evaluation will be carried out to analyse the results of the experiments. It will compare the efficiency of Hadoop and Nephele, referring to data obtained in the experimentation phase. The results will be compared to existing research (obtained during the background research) and the hypothesis, and any discrepancies will be discussed.

An evaluation will also be carried out to determine whether the aims and objectives of the project have been met. This will involve determining whether the research questions have been answered in a satisfactory manner, and determining the relevance of the project in regards to existing literature and how it contributes to the research landscape. Any future scope for research will be identified.

The methodology and management of the project will also be evaluated to identify areas of improvement in project management.

1.4 Requirements

The projects success will be judged on whether it meets a set of requirements. Requirements will be broken into 2 categories. Minimum requirements are essential for the success of the project. Failure to meet any of the minimum requirements would seriously compromise the project, and would make it unlikely to qualify for a pass grade. There are also various extended requirements. These requirements can be seen as ‘stretch goals’. They are not essential for the project, but will contribute considerably to the end product.

1.4.1 Minimum Requirements

By meeting the minimum requirements, it should be possible to replicate the results in existing literature [40].

- Install Hadoop

  Installing Hadoop is a necessary part of the project as it must be used to run data processing experiments. At a minimum, Hadoop should be installed on the University of Leeds School of Computing Cloud Testbed.

- Install Stratosphere

Installing Stratosphere is a necessary part of the project so experiments can be run, and the results compared with Hadoop. At a minimum, Stratosphere should be installed on the University of Leeds School of Computing Cloud Testbed.

- Run a MapReduce processing task

  There are various basic MapReduce tasks which are provided as examples for both Stratosphere and Hadoop. Common examples are Word Counting and the Terasort benchmark. The example code provided could be executed against the same dataset to provide a basic comparison of the performance of Hadoop and Stratosphere in a MapReduce situation.

- Run a non-MapReduce processing task

  There are various pieces of example code for Stratosphere which demonstrate non-MapReduce processes such as K-Means clustering. There are also various libraries and frameworks for Hadoop which provide implementations of algorithms such as K-Means. These implementations can be compared in order to determine whether PACT overcomes the issues with MapReduce.

- Run an experiment on a different cluster size

  In order to determine how scalable the solutions are, an experiment must be run on more than one cluster size. As a minimum requirement, the MapReduce benchmark (e.g. Word Count, Terasort) could be run on 2 and 4 machines for each processing technology, with the results compared for scalability.

1.4.2 Extended Requirements

The extended requirements provide extensions to the project which constitute work that furthers the existing research.

- Real-world problem

  In addition to running a basic MapReduce and non-MapReduce problem, a ‘real-world’ scenario could be chosen. This would reflect the use of data processing tools in a more natural way, rather than relying on synthetic benchmarks. There are various datasets made publicly available [7], and these could be analysed using both of the data processing tools with their execution times compared. An example of the type of problem which may be relevant is the 1000 genome project [22].

  In addition to testing the performance of Hadoop and Stratosphere in a less artificial environment, this extended requirement will also require new code to be developed to run on the processing technologies, rather than using existing example code.

- Scale all experiments

  Rather than just scaling one experiment across 2 different cluster sizes, all experiments could be ran across different cluster sizes to provide a more comprehensive measure of scalability. Additionally, the cluster size could vary to a larger extent. Rather than running experiments on 2 cluster sizes, they could be executed on a larger range of machines.

- Public Cloud

  As a minimum requirement, the clusters will be built on the University of Leeds School of Computing Cloud Testbed. An extension would be to run experiments using a public Infrastructure as a Cloud service such as Amazon EC2 or Windows Azure.

1.5 Schedule

The project schedule will be created around a set of fixed milestones in the project process. Work will be allocated in weekly sprints to ensure that the relevant work is completed in time for each milestone. Work will be prioritised so that scope can be cut to meet a milestone if required.

Figure 1.1 is a Gantt chart representing the predicted schedule for the project. At this stage it is left intentionally vague, as more information will become available as the initial stages of the project are completed. For example, there is time allocated for Experiment Implementation and Additional Requirements, rather than actual individual experiments or work required to complete specific additional requirements.

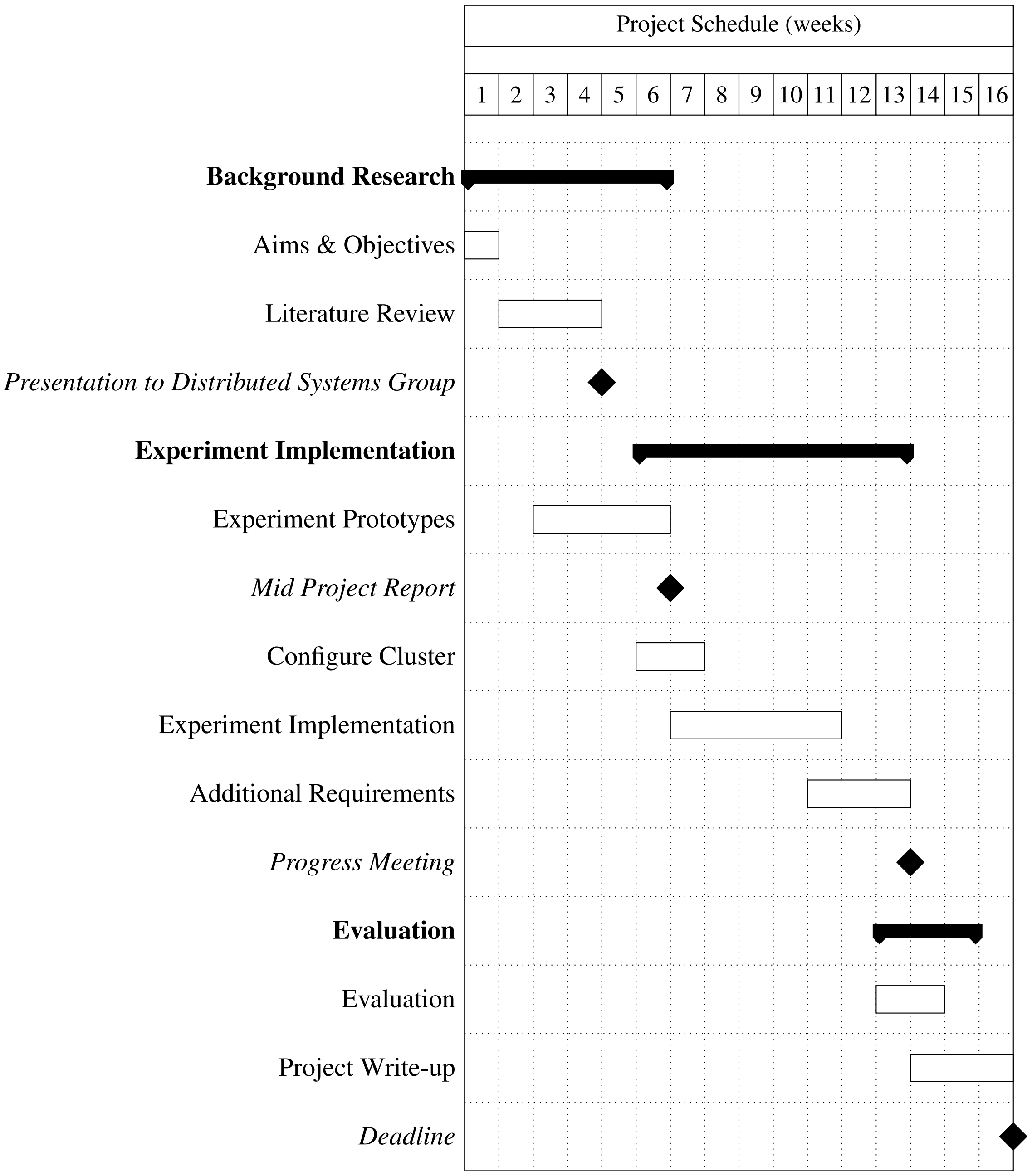

Figure 1.1: Gantt Chart of Project Schedule

1.5.1 Milestones

There are a series of milestones which must be met throughout the project. This provides a framework for the project plan, as necessary tasks must be completed before the milestones.

1. Submission of Aims & Objectives

   The Aims & Objectives of the project must be decided, which is crucial in setting the scope and direction for the rest of the project.

2. Presentation to the Distributed Systems and Services Group

   The presentation to the Distributed Systems and Services group provides an opportunity to get early feedback on the project, and present much of what will be present in the mid year report. In order to present the project, a sufficient amount of content must be prepared. By this point a substantial amount of background research should have been carried out, and early ideas about experimental design should be formed.

3. Mid Project Report

   The Mid Project Report is a chance to get feedback from the assessor about the direction and progress of the project. By this stage the background research should be completed, and should demonstrate that necessary steps have been taken to understand the problem. The Aims & Objectives should be clearly defined, with a set of deliverables, a methodology, and a schedule. At this point experimental design will not have been completed, but the Mid-Year Report should give indications about the type of experiments that will be executed and why they were chosen. Finally, evidence of basic implementation should be shown, to demonstrate knowledge of the tools required for experiment implementation.

4. Progress Meeting

   The progress meeting occurs in the late stages of the project, so at this point most of the work should be completed. Experiments should have been implemented and performed, and code should be available for inspection.

5. Final Report Submission

   The final report submission marks a point where all work on the project has to be complete.

1.6 Deliverables

The project will deliver 2 things.

1. An evaluation of the efficiency of Hadoop and Nephele in various data processing situations

2. Code for experiments, designed to test the efficiency of Hadoop and Nephele

Chapter 2

Background Research

This chapter is intended to give an overview of the current research landscape, and to summarise the core technologies and concepts which form a basis for this project. The chapter will discuss trends toward Cloud Computing, how it is relevant to Data Processing, and key technologies which have been developed in this area, giving a foundation for further investigation into the efficiency of Data Processing techniques and how they are impacted by the Cloud.

2.1 Cloud Computing

Cloud Computing is the latest major infrastructure paradigm which looks to deliver on the promise of Utility Computing. In practice, the term 'Cloud Computing' is ambiguous. Whilst no clear definition exists many experts agree that the cloud exhibits the core benefits of Utility Computing such as elasticity and scalability, whilst making heavy use of virtualisation and pay-per-usage business models [38].

Elasticity in clouds refers to the ability for a user to dynamically select the amount of computing resources they require, allowing them to scale their applications according to demand. The resources that can be acquired are essentially limitless from the user's perspective [26].

Elasticity represents a dramatic shift from the 'traditional' method of building and deploying applications. Rather than purchasing and provisioning hardware and acquiring physical space (such as a data centre), a user can use a Cloud Service Provider. This allows for greater flexibility in business and application development, as users can cope with unpredictable or inconsistent levels of demand. An example of where this flexibility would benefit a company is in the case of an online store. A store may receive fluctuating traffic throughout a year, such as being particularly busy around the Christmas period. It would be economically inefficient for the store to purchase extra servers to cope with demand over the festive period, as they would be redundant for the majority of the year, but they must improve hardware capability in order to take advantage of the extra business. The inherent flexibility from the elasticity of the cloud would allow the store to simply acquire new computing capacity from their Cloud Service Provider on a temporary basis, giving them the capability of accommodating with the peak traffic but not using (or paying for) the extra resources when they are not needed. Having this scalability is seen as a core benefit of Cloud Computing.

A Cloud Service Provider is an organisation that provides access to Cloud Computing resources. They manage the underlying hardware, and typically provide APIs and other methods for a user to manage their resources. Some of the largest Cloud Service Providers are Amazon through AWS (Amazon Web Services), Microsoft through Windows Azure and Google through Google Apps.

A Cloud Service Provider does not have to be an external organisation, but when they are they typically use pay-per-usage business models. Rather than pay a fixed monthly cost, or have a one-off license fee, customers will pay the Cloud Service Provider for the resources they use (usually on a per-hour basis). For example, a 'Medium' size Virtual Machine costs 0.077 an hour from Windows Azure [27]. This allows businesses to only pay for the resources that they need.

2.1.1 Related Ideas

2.1.1.1 Utility Computing

Utility Computing is the idea that households and businesses could outsource their demand to external companies, who provide the relevant amount of service on a pay-per-usage basis. Customers would access computing resources over a network, and would pay for the length of computing time that they use.

This is analogous to other utilities such as Gas or Electricity. In the case of Electricity, power is provided from the National Grid and the customer pays for how much they use. This allows a customer change the amount of power they require without having to pay a fixed cost (for example, using less electricity when they are on holiday).

Utility Computing is an established concept, with leading thinkers such as Leonard Klienrock (part of the original ARPANET project) referencing it as early as 1969 [24]. Various technologies have emerged which offer some attributes associated with utility computing, with Grids and Clouds appearing to be the most promising [15].

2.1.1.2 Virtualisation

Virtualisation is one of the key enabling technologies behind Cloud Computing. It abstracts away the details of the physical hardware and allows Cloud Service Operators to run several virtual machines on one physical machine [46], completely independently of one another. This allows for customers applications and the physical hardware to be consolidated, utilising resources more efficiently and making it financially feasible to run a cloud [19].

The configurability afforded by virtualisation is another property essential to Cloud Computing. It allows Cloud Service Providers to support a diverse range of applications, which may have different requirements (high compute, high memory, etc). Virtualisation allows this to be achieved, as it would be prohibitively costly at hardware level [19].

The reliability of a cloud can also be improved through the use of virtualisation techniques. Virtual Machines can be backed up, migrated and replicated simply, allowing applications to recover from hardware failure.

2.2 Cloud Computing Service Models

Depending on the scenario, the Cloud offers several different service models. These models allow for clients to provision services in a different manner depending their requirements.

The different service models provide different levels of abstraction for the user. In Infrastructure as a Service, the user has full control over the machines that they acquire from the Cloud Service Provider, where in Software as a Service they are given less control, and need not worry about the underlying hardware whatsoever.

2.2.1 Infrastructure as a Service

Infrastructure as a Service (IaaS) provides an abstraction on top of a virtualisation platform, so that the client does not need to worry about what method of virtualisation is being used, and does not have to learn about the underlying technologies [9].

Clients can request Virtual Machines in varying configurations, and a Virtual Infrastructure Manager will provision an appropriate Virtual Machine on a physical machine which has capacity. In addition to allowing users to provision virtual machines, IaaS systems may allow a user to configure other infrastructure elements, such as virtual networks.

This provides a great deal of control to the user, as they are essentially renting a machine of a requested specification for a short period of time. They are free to install whatever Operating System and software on the machine as required, and can configure it in essentially any way.

An example of an Infrastructure as a Service provider would be Amazon EC2 [6]. Amazon provide a variety of different Virtual Machine types, including those specialising in High Performance Computing or applications requiring a large amount of memory. Virtual Machines can use a range of images provided by Amazon (including Windows and various distributions of Linux), or users can create and upload their own custom Virtual Machine images.

2.2.2 Platform as a Service

Platform as a Service (PaaS) is a higher level abstraction which allows applications to be built and deployed without worrying about the underlying Operating System or runtime environment [23]. The user still specifies the resources required, but no longer has to manually manage the virtual machines.

The Cloud Service Provider will maintain the machines, providing the necessary software (Operating Systems, Web Servers, etc) and updating them frequently.

The advantage of PaaS is that it allows users to deploy their own applications, without having to worry about maintaining the underlying infrastructure. Whilst this decreases the control the user has over the deployment environment, it reduces the complexity of managing the infrastructure themselves.

PaaS offerings may also provide supplementary services to users, such as health and availability monitoring, or auto-scaling.

Windows Azure is an example of a Platform as a Service provider [28]. Whilst they provide Infrastructure as a Service offerings, they also provide Platform as a Service capabilities through Windows Azure Web Sites. Windows Azure Web Sites allow users to upload applications written in a variety of web technologies (ASP.NET, Python, PHP, Node.js) and have them hosted in the Windows Azure runtime environment. This means the client does not manually have to manage web servers, frameworks and other necessary technologies.

2.2.3 Software as a Service

Software as a Service (SaaS) refers to providing access to applications over the internet on-demand [46]. Software is centrally hosted by the Cloud Service Provider, and clients can access the application through a web browser or other form of client. As the software is centrally hosted, Cloud Service Providers can handle updating the software for all users, ensuring all users benefit from bug fixes or additional features.

Software as a Service applications can reduce the cost of deploying and using software for an organisation as they don't have to purchase their own hardware, install and configure software, and can avoid having technical support staff. An example of a successful Software as a Service application is Salesforce [33]. Salesforce is a Customer Relationship Management tool which charges organisations per user, making it a viable choice for small businesses. Salesforce can be accessed through a web browser, enabling customers to use their software regardless of location or device.

2.3 Cloud Computing Deployment Models

Cloud Services can be deployed in several different ways. The National Institute of Science and Technology defines 4 Cloud Computing deployment models [26].

2.3.1 Public Cloud

A public cloud is designed to for use by the general public. A public cloud will typically be owned by a third-party Cloud Service Provider such as Microsoft or Amazon, and will serve lots of different individuals and organisations.

The benefit of a public cloud is that physical infrastructure is completely managed and maintained by the Cloud Service Provider, reducing the effort required to provision computing resources and therefore maximising the benefit of the cloud.

2.3.2 Private Cloud

A private cloud is designed for use by one organisation. This allows individual components of an organisation (such as departments, product groups or engineering teams) to utilise the benefits of Cloud Computing, whilst allowing the organisation to maintain control over the computing resources. A private cloud allows an organisation to completely tailor the cloud to their unique requirements, including specialised hardware, specific Platform/Software as a Service offerings and control over the permissions clients of the cloud environment have.

A private cloud can also be necessary in overcoming concerns about security and data governance. One of the major disadvantages of a public cloud is a lack of control over data (real or perceived). It may not be possible to ascertain where data is geographically located (and therefore what legislation applies to it), and companies may simply not trust confidential information with a third-party Cloud Service Provider.

Whilst a private cloud still requires an organisation to acquire, provision and manage hardware (rather than outsourcing to a third-party Cloud Service Provider), it can still provide a reduction in cost over time. A private cloud allows centralisation of an organisations computing resources, allowing components of the organisation to use resources elastically as required from the central private cloud, rather than duplicating necessary computing infrastructure.

2.3.3 Community Cloud

A community cloud is developed by several organisations that have shared requirements. It allows them to pool together computing resources to the benefit of all organisations involved, reducing the investment needed for a private cloud. Organisations may opt to develop a community cloud to reduce dependency on a public cloud provider, mitigating the privacy and security concerns often cited as being a problem for public clouds, and allowing the resulting cloud system to be tailored to take in to account legislative or administrative restrictions which might be part of some industries or other organisational groups.

Whilst creating a community cloud provides tangible security and customisability benefits over a public cloud, and cost benefits over developing a private cloud system, they also have their issues. Community clouds can make it more difficult to deal with standard distributing computing issues such as latency and resource management, and additional security requirements may be needed [14].

2.3.4 Hybrid Cloud

A hybrid cloud is a composition of 2 other cloud types (public, private and community). Typically, a hybrid cloud uses technology to enable data and applications to be transported between clouds. An example of a hybrid cloud solution would be a private cloud, which utilises a public cloud if it runs out of resources. This allows the private cloud to exhibit the same elasticity benefits of a public cloud, appearing to clients as though it has essentially limitless resources (whilst actually just making use of a public cloud for overflow capacity).

It may be difficult to create a hybrid cloud in some situations as the technologies used may be complete distinct and difficult to combine. There may also be concerns about data security and privacy. Business rules may have to be defined which specify (for example) what types of data can be stored on a public cloud, and what types of data are too sensitive and must remain private at all costs.

2.4 Big Data

The term 'Big Data' refers to large scale data sets which present new problems for the processing and analysis of data. These datasets are often too large to be comfortably worked with using standard tools for statistical analysis [35]. Sources of Big Data are diverse; datasets can be from science, industry, social networking sites or various other sources. Consider the following:

- In 2011, Facebook had 30 PetaBytes of data used for performing analytics [44].
- The Large Hadron Collider generates 25 PetaBytes of data annually [43].
- Phase one of the 1000 Genome Project has a dataset of 180 TeraBytes of human genetic information [22].
- In 2008, Google were capable of processing 20 PetaBytes of data a day [18].

2.4.1 Challenges

Big Data introduces several challenges which make the use of traditional analysis tools and techniques infeasible.

2.4.1.1 Scale

The defining characteristic of Big Data problems is their massive scale. Typical problems may require processing datasets in the magnitude of PetaBytes of information. This makes it unrealistic to process the data in serial, as it would take far too long. Data processing techniques designed for Big Data problems must process information in parallel.

The scale of the data also makes storing data on a single machine unrealistic. Whilst the capacity of hard drives has been increasing over time, the speed of I/O has not increased by the same factor [41]. This makes it time consuming to read an entire dataset off of one physical disk, even if a disk exists with enough capacity.

Unfortunately, the alternative approach of distributing data across multiple machines also has issues. Using a higher number of nodes increases the likelihood that one of those nodes will fail, which could lead to irrecoverable data loss. In order to prevent this, systems for processing Big Data should be designed to be fault tolerant, usually by introducing redundancy.

2.4.1.2 Predictability

Not all Big Data is static. There has been increasing interest in performing sentiment analysis on social networks to predict the results of real-world events such as box office revenues [10], financial markets [12], or surveys on political opinion [30].

It can be difficult to predict the quantity of data which may need to be processed when analysing real-time data, such as that gathered by Twitter. In August 2013 Twitter experienced an unexpected increase in traffic, leading to a new record for the number of tweets in a second [25]. This highlights the potentially unpredictable nature of real-time data analytics.

The challenges of predicting resources required for real time data analytics show the advantage of using Cloud Computing for data processing. The elastic nature of clouds would allow resources to be dynamically acquired when an unpredictable increase in data occurs.

2.4.1.3 Data Heterogeneity

It is very rare for different data sources represent data in the same way. Where data is processed from multiple sources, the lack of standard methods of storing data can make it difficult to develop accurate processing techniques [16].

A lack of structure in data can also cause problems with analysis. Much of the data used in large scale processing is strongly unstructured (social network data, text from articles, etc) which can cause issues with processing techniques. For example a simple data processing task which counts the frequency of words in a text document may produce inaccurate results if there are spelling mistakes in the corpus.

2.5 MapReduce

MapReduce is a programming paradigm designed as a generalised solution to processing large scale datasets [17]. A programmer specifies both a *map* and *reduce* function, a set of input data, and a location for the output. A MapReduce runtime determines how to distribute the task, taking the data and dispersing it amongst its nodes, enabling the data to be processed in parallel.

The key advantage of MapReduce is that it provides an abstraction on top of concurrent programming. This allows a developer to specify what they want to do to the data (in terms of the *map* and *reduce* functions) without having the obscuring the code and having the cognitive overhead of handling common concurrency issues, such as fault tolerance, data distribution, and safe parallelism without the risk of problems such as race conditions and deadlocks.

The MapReduce method of performing distributed parallel data processing is inspired by the *map* and *reduce* functions present in many functional programming languages [17].

2.5.1 Map

The first function that a developer provides is the *map* function.

$$map = (k, v) \rightarrow list(k', v')$$

The *map* function receives a key and a value as input (an *Input Pair*), and returns a list of (different) keys and values (a list of *Output Pairs*. The MapReduce runtime will take all output pairs with the same key, and combine them in to a list before passing the resulting values to the *reduce* function. The reduce function therefore receives an argument of type $(k, list(v))$.

2.5.2 Reduce

The *reduce* function is the second function provided by the developer.

$$reduce = (k, list(v)) \rightarrow list(v')$$

A *reduce* function merges values together to produce a smaller list of values. Typically, the reduce function is an aggregate function, returning either 1 or 0 results (such as sum, average, etc).

2.5.3 Word Counting Example

A common MapReduce operation is word counting. The following pseudo-code implementation of the *map* and *reduce* functions for counting words in a document obtained from [17].

Algorithm 1 Map function for word counting

```
Require: key: document name
Require: value: document contents
  for all w in value do
    EmitIntermediate(w, 1)
  end for
```

The *map* function takes a document, and returns a number of Key/Value pairs, where the key is a word appearing in the document, and the value is 1. The runtime environment will then cluster the values together where the keys match. Each word will therefore have a list of 1's, with the length of the list indicating how many times the word has occurred.

Algorithm 2 Reduce function for word counting

```
Require: key: a word
Require: value: a list of counts
  result = 0
  for all v in values do
    result += v
  end for
  return result
```

The reduce function takes the list of values (all of which are 1) and sums them together. This gives a total number of times the word has occurred in the document.

2.5.4 Disadvantages

The primary criticism of MapReduce is that it provides a limited programming framework [45]. Whilst the MapReduce paradigm is a relatively general concept which applies to many types of application, not all problems can be formulated in a way which is compatible with MapReduce. The MapReduce process consists of a static *map* operation, performed across the data, and a static *reduce* operation performed subsequently. The *map* and *reduce* functions are both stateless.

This imposes limitations on the types of problem which can be solved by MapReduce. If it is not possible to formulate a *map* and *reduce* function for a problem, it cannot be solved by the MapReduce paradigm. Alternatively, whilst it may be possible to express a data processing task in terms of *map* and *reduce*, it does not always provide the most fluent or efficient method of solving the problem.

Prominent examples of this type of algorithm can be found in Machine Learning, where problems are often iterative in nature. Iterative tasks, such as K-Means Clustering can be implemented using the MapReduce paradigm [47], but often require multiple passes of the *map* and *reduce* functions, which causes a significant performance penalty as data must be loaded from the disk every iteration.

2.5.5 Hadoop

Hadoop is an Open Source implementation of the MapReduce programming paradigm [3]. It consists of several modules designed to overcome the problems associated with large scale data processing. In addition to the MapReduce programming model (Hadoop MapReduce), the Hadoop platform also has a distributed file system (HDFS) and a resource management and scheduling component (Hadoop YARN).

Hadoop is a particularly popular implementation of the MapReduce interface in the data processing community, emerging as the de-facto standard implementation of MapReduce [32].

2.6 PACTs

Parallelization **C**ontrac**t**s (PACTs) is an alternative programming paradigm which is a generalization of the MapReduce concept [11]. The aim of PACT is to overcome the perceived weaknesses in the MapReduce paradigm, whilst still providing a generic enough framework to represent any data processing task. PACT sets out to improve MapReduce by observing that the *map* and *reduce* functions alone cannot represent all data processing tasks in a natural or efficient way, that MapReduce does not provide flexibility in the order of operations (strictly a *map* followed by a *reduce*) and that MapReduce makes no assumption about the user defined functions, limiting the scope of potential optimizations.

PACT aims to solve these issues in 2 ways. It introduces the concept of Parallelization Contracts, which define properties of a user defined functions input and output data, and it represents the data flow of a data processing job as a Directed Acyclic Graph (DAG), rather than strictly as a *map* and *reduce* task.

2.6.1 Processing Tasks

A data processing job can be described as a set of tasks, where each task performs a transformation on the data. A processing task has 3 core components.

A data processing task acts upon a list of records. Unlike the classic MapReduce paradigm, input is not restricted to Key/Value pairs. A record can consist of any number of fields, each which can be either a key or value. All fields must be capable of being serialized and deserialized, and fields defined as keys must be capable of being compared with other keys.

2.6.1.1 Input Contract

A task can be described as consisting of multiple Parallelization Units (PUs). A PU is a core block of computation which must be performed in serial. PUs are isolated from one another, and do not require communication, meaning that they can be distributed across different nodes and can be ran in parallel to one another. PACT specifies a set of Input Contracts which define how a problem is broken down into PUs. The Input Contract is therefore key in determining how the problem is parallelized and distributed between nodes.

The PACT Programming Model provides 5 Input Contracts [4].

- **Map** takes a list of records and creates a PU for each individual record. This allows every record to be processed in parallel. The Map Input Contract is typically used for transforming or filtering data.

- **Reduce** takes a list of records and groups them by record key. It then creates a PU for each group, allowing the resulting subsets to be processed in parallel. The Reduce Input Contract is often used for aggregating data, performing functions like sum or average.

- **Cross** is a multi-input contract, taking 2 lists of records. It takes the Cartesian product of the 2 lists, and creates a PU for each element in the resulting set. This allows every combination of input to be processed in parallel.

- **CoGroup** takes 2 lists of records and groups them by key. A PU is then generated for every resulting group. This is similar to the Reduce Input Contract, but it operates across 2 lists rather than 1.

- **Match/Join** takes 2 lists and matches records from the first with records from the second based on whether they have the same key. A PU is generated for every resulting pair of records. This is conceptually similar to an equi-join operation.

2.6.1.2 User Defined Function

The User Function is the programmer defined function which will be executed on the data. This may be counting the frequency of words, searching for a particular string, or various other things. The arguments that the User Function will receive are based on the input contract - if a Map Input Contract is used, the User Function will receive an individual record. If a Reduce Contract is used, the User Function will receive a group of records with similar keys.

2.6.1.3 Output Contracts

An Output Contract is an optional component of a data processing task. An Output Contract is used to provide guarantees about the type of data that the user function will return. This allows the PACT runtime to optimise the process to make a more efficient data flow. An Output Contract can indicate that data does not have to be repartitioned between nodes before going on to the next stage data processing task.

There are various Output Contracts.

- **Same-Key** ensures that each record generated by the user defined function has the same key as its input. This ensures that any partitioning between nodes can be kept for the next stage of processing.

- **Super-Key** provides a super-key of the records it was generated from. A function with the Super-Key output contract allows partitioning between nodes to be kept the same.

- **Unique-Key** gives each record a globally unique key across all nodes. This allows records to be stored in data sources which require globally unique keys (e.g. in a relational database table with a primary key).

- **Partitioned-By-Key** partitions records by their key. This has similar implications to the Super-Key contract, in that partitioning is kept, but there is no order within the partitions.

2.6.2 Directed Acyclic Graph

Data Processing jobs are represented as Directed Acyclic Graphs [39]. A vertex in the graph represents an individual task (a combination of an input contract, a user defined function, and optionally an output contract). Edges represent the communication between different tasks. Tasks can be connected to multiple other tasks.

The primary advantage of a DAG is that it allows for a more flexible representation of data processing tasks, allowing complex workflows of tasks, rather than being limited to a map task, followed by a reduce task. This allows for arbitrarily complex graphs to be defined, allowing more complex data processing jobs to be completed.

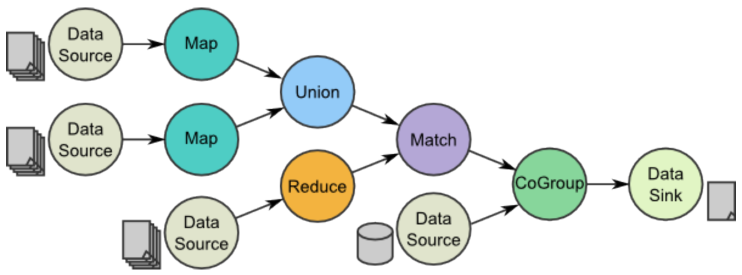

Figure 2.1: An example of a complex dataflow [2]

Directed Acyclic Graphs can also be composed together, by connecting the output of one graph to the input of the next.

The classic MapReduce paradigm could be emulated by having 2 tasks, one with the Map Input Contract and another with the Reduce Input Contract (neither would have Output Contracts). The result of the Map task would be piped directly in to the Reduce task.

2.6.3 Nephele

Nephele is the reference implementation of the PACT programming model. It is a runtime environment which takes a Directed Acyclic Graph specifying the desired data processing workflow (the Job Graph) and and compiles it in to an Execution Graph - a more concrete description with information about how the tasks will be parallelized and what types of computing resources will be used [40].

A secondary focus of the Nephele runtime is to take advantage of the opportunities afforded by Cloud Computing. The core benefit of Cloud Computing is elasticity, allowing resources to be obtained on demand. By removing the assumption that resources must be static throughout the duration of a data processing job, Nephele is able to utilise the dynamic resource allocation made possible by the cloud. This changes the emphasis of task scheduling, allowing the scheduler to determine and acquire what resources a given task needs, rather than attempting to schedule tasks depending on the resources available at the time [40].

Chapter 3

Experiment Design

A series of experiments have been designed in order to objectively analyse the efficiency of Nephele and Hadoop. This section discusses the design of the experiments in regards to the aims of the project.

Experiments are designed in a modular fashion, enabling a subset of them to be completed whilst still providing relevant research results. This approach was taken to limit the risk that is introduced by the time constraints of the project.

3.1 Experiment Variables

There are a total of 30 experiments with 3 varying factors, in order to test the data processing tools in a range of relevant scenarios.

3.1.1 Data Processing Tools

The first factor that will be varied is the tool used to process the data. This relates to the first two research goals: providing a comparison of the MapReduce and PACT programming paradigms, and ascertaining how well Nephele can perform MapReduce tasks.

3.1.1.1 Hadoop

Hadoop is an open source implementation of the MapReduce paradigm, which has become the de-facto standard for big data processing [32]. This makes it an ideal choice for comparing MapReduce and PACT.

Experiments will be run in Hadoop to provide an indication of the efficiency for the current state of the art in data processing. This provides a reference point to which alternative data processing technologies can be compared.

3.1.1.2 Nephele (PACT)

Nephele is a tool which implements the PACT programming paradigm, which as a superset of MapReduce can also perform tasks in a MapReduce style. This means that experiments can be implemented using the same tool, but in different programming paradigms, allowing a fair test of PACT vs MapReduce. This is designed to answer the first research question, namely whether or not PACT overcomes the inherent disadvantages of the MapReduce paradigm.

In a broader sense, it allows a complete comparison of Hadoop and Nephele. In real-world data processing scenarios, users of Nephele would be unlikely to restrict their programming options to just MapReduce, and would be likely to use other aspects of PACT as appropriate. Running experiments in the same way allows a fairer comparison of the 2 tools, where all of their functionality is available.

3.1.1.3 Nephele (MapReduce)

Experiments will also be run using just the Map and Reduce operations of Nephele. Whilst Nephele has a wider set of operations, solving data processing tasks using just Map and Reduce allows Nephele and Hadoop to be directly compared, aiming to answer the second research question. If Nephele can perform MapReduce tasks with comparable efficiency to Hadoop, it will indicate that Nephele is a more efficient tool for large-scale data processing in general, not just for those situations where a problem cannot be formulated in terms of MapReduce.

3.1.2 Data Processing Task

The second varying factor will be the data processing task that the tools have to complete. The tasks have been chosen to represent real-world issues as closely as possible, to give an indication of performance in scenarios where the tools are likely to be used, rather than having synthetic benchmarks such as word counting or Terasort.

3.1.2.1 Reverse Link Graph

The first problem the data processing tools will have to solve is generating a Reverse Link Graph [18]. In some situations, finding the transpose of a directed graph can be difficult. For example, when considering the Web Graph; it is relatively trivial to find the edges coming out of a vertex (as the hyperlinks on a page indicate its outgoing edges), but it is far more difficult to find incoming edges (pages which link to the current page).

The Reverse Link Graph problem has been solved as it is a ‘classic’ MapReduce problem. A ‘classic’ MapReduce problem is a problem which can easily be formulated in terms of MapReduce. Classic MapReduce problems tend to be stateless, and do not depend on other parts of the data. This makes it easier to compute the solution in parallel, but means that many problems are not well suited to MapReduce (such as naturally iterative problems). An example of a classic MapReduce problem would be Word Counting, as each document can be processed in parallel.

The Reverse Link Graph has been chosen in order to measure how effective both Hadoop and Nephele are at processing data using the MapReduce paradigm. This specifically relates to the second research question ("How well does Nephele perform MapReduce tasks?").

Finding a reverse link graph of the web is a key component in many search engines [31] where backlinks are used to indicate how important a page is, and can also be used to analyse a network of academic articles by treating references as edges [21].

3.1.2.2 PageRank

The second problem for the data processing tools to tackle is the PageRank algorithm. PageRank is a method of objectively and mechanically rating the importance of a web page, trying to measure human interest and the attention paid to the page [31].

The PageRank algorithm is an example of the type of problem that MapReduce is traditionally cited as being poor at completing. It is an iterative algorithm which converges on a PageRank for each node. It requires information from adjacent nodes to compute a nodes PageRank at each step, which means that each step of the iteration must be performed sequentially, as part of a separate MapReduce pass.

By contrast, PageRank should use many of the additional features that the PACT programming model provides over MapReduce. The presence of iteration support and dedicated operations for Joins should test the additional operations that PACT provides in order to determine whether they yield tangible performance benefits.

Computing the PageRank of a graph of pages is a suitable test of how well the PACT programming model overcomes the issues of MapReduce, as it exercises many of the additional features. This will provide an answer to the first research question ("Does PACT overcome the inherent disadvantages of the MapReduce paradigm?").

3.1.3 Cluster Size

The size of the cluster will be varied in order to give an indication of how well the data processing technologies perform at scale. This is a key factor in processing large scale data, as technologies which do not scale well as machines are added are unlikely to perform well in real-world scenarios, where the size of data may run into the Petabytes.

The elasticity of the Cloud will be required to scale the size of the cluster for different experiments. Specifically, experiments will be run using 2, 4, 6, 8 and 10 nodes. Scaling the size of the cluster will test both the scalability of the data processing tools, and the ability for the tools to harness the elasticity of the cloud, meeting the third research aim.

3.2 Measurements

Measurements will be taken from the experiments in order to give quantitative metrics which can be used for objective analysis.

3.2.1 Runtime

Runtime is the length of time that a program (or data processing task) takes to execute. Runtime is one of the primary mechanisms for determining the efficiency of the data processing solutions. The time it takes to perform a task is a key element in large scale data processing, as the sheer scale of data can cause operations to take a long period of time.

3.2.2 Scalability

The Scalability of data processing tools is also a key metric in determining how efficient they are. The scalability of a system refers to how well it accommodates additional resources [13], such as additional machines.

In the context of Data Processing tools such as Hadoop and Nephele, it is desirable for systems to be able to scale according to both the size of data, and the number of nodes in the cluster. A data processing tool should be able to handle any amount of data, providing that it has the correct amount of system resources. Indeed, this is what distinguishes tools like Hadoop from other data processing tools such as numpy - the ability to analyse petabytes of data. A Data Processing system should also be able to scale according to the number of nodes added. There should be no upper limit to the number of nodes in a cluster, as this provides an important mechanism of adding the necessary resources to handle larger processing tasks. Adding a large number of nodes with lesser resources may be more cost efficient than adding fewer nodes which have comparatively larger system resources. This may be a reason why many users that engage in large scale data processing build clusters from commodity components [36].

The specific type of scalability that will be considered in these experiments is Runtime Scalability. Intuitively, as nodes are added to the cluster the runtime for experiments should decrease as work can be partitioned out to a larger set of working nodes. In an ideal world, this would follow a linear progression; as the number of processors is doubled the runtime should be halved. This can be measured by taking the runtime of a task when the processor count doubles (so the processor count is 2^p where p is an integer representing the processor count) and taking the derivative.

3.2.3 Alternative Measurements

Various alternative mechanisms could be used to determine the efficiency of the data processing tools. The scope of the project will be limited to measuring the runtime and runtime scalability, but some other measurements which could be used are as follows.

3.2.3.1 Data Scalability

It is important for clusters to be able to deal with large amounts of data in a reasonably efficient manner. A further test of scalability would be to test the cluster with various different datasets of different sizes. As with runtime scalability, this would ideally be linear, so that when the size of the data is doubled, runtime would also approximately double. The data processing tools may be of less use if the runtime increase is larger (for example, an exponential increase in runtime) as this would make running tasks on petabytes of data unreasonable.

3.2.3.2 Resource Utilisation

The resources used on nodes can be measured to determine how efficient the data processing tool is. An efficient data processing tool could be defined as one which uses the resources of each node effectively, by utilising a high amount of memory and CPU capacity. This would indicate that the data processing tool is using each node to the highest capacity possible, rather than requiring extra nodes to be added for more resources to become available.

The utilisation of the network can also be used in determining efficiency, as a significant factor in how well a system scales is how much it communicates with other nodes. Larger amounts of communication overhead will increase the runtime of a system as it scales. Measuring the network overhead gives an indication of how much nodes communicate with each other, therefore giving an indication of communication overhead and scalability.

3.2.3.3 Cost

Another metric which can be used to compare data processing tools is cost efficiency. In situations where a cluster is being built using Cloud Computing services, running data processing jobs will cost on a per-usage basis. Depending on the runtime scalability, it may be more cost efficient to run a job on a larger number of nodes (so that it completes faster), versus running it on a smaller number of nodes (which will take longer, but cost less per hour).

It may also be more cost efficient to create a private cluster of machines to run data processing jobs, rather than using a pay-per-usage cloud model.

3.3 Hypothesis

The project as a whole focuses on answering 3 core research questions. Rather than hypothesising the result to each individual experiment, there will be a prediction for each research question. Each experiment is used as a way of gathering information to answer one of the three core questions, and as such the results to individual experiments can be inferred from the broader descriptions.

3.3.1 Does PACT overcome the inherent disadvantages of the MapReduce paradigm?

In order to determine whether PACT overcomes the inherent disadvantages of MapReduce, both tools will solve the PageRank problem; a data processing task which exhibits many of the features generally considered to be a poor fit for MapReduce.

It is likely that PACT will demonstrate a clear superiority over MapReduce in these situations by having a smaller runtime, regardless of the number of nodes that are used to process the data. The presence of dedicated PACT operators for both iteration and delta-iteration will be a significant benefit for PACT, as PageRank is a naturally iterative algorithm. Additionally, PACT provides operators for joins, which are used throughout the PageRank algorithm. This will allow the Stratosphere runtime to optimise these steps accordingly.

In contrast, MapReduce lacks a dedicated method of iteration, meaning that any iterative algorithm must be implemented using multiple Map and Reduce steps. Additionally, any joins required by the algorithm will have to be implemented using additional Map and Reduce operations, causing multiple steps per iteration.

This will cause the PACT runtime to outperform the MapReduce runtime when running the PageRank experiment, indicating that it overcomes the inherent flaws in the MapReduce paradigm.

3.3.2 How well does Nephele perform MapReduce tasks?

The Reverse Link Graph problem will be used to determine how well Nephele performs MapReduce tasks. As a classic MapReduce problem it allows comparison with Hadoop, the current status quo of data processing tools.

In these situations Hadoop should be the clear winner in terms of performance. It is a simpler tool, designed for running MapReduce processes without the overhead associated with having other operations available. This allows the MapReduce runtime to be highly optimised to performing those tasks. As a research project, it is far more likely that the limited development resources of Nephele would be focussed towards optimising other operations, designed to be the source of an advantage over MapReduce. This may lead to a less efficient MapReduce implementation.

Nephele is also a relatively immature project in comparison to Hadoop. It has existed for far less time, and therefore has less contributions. It is a less visible project than Hadoop, and has been used less in production. On this basis it can be reasoned that Nephele will have had less development attention towards optimising the Map and Reduce processes, and will therefore have inferior performance with these tasks.

3.3.3 How do Hadoop and Nephele perform in a highly elastic Cloud Computing environment?

An important factor in how suitable data processing tools are to large scale data sets is how well they scale. This is particularly important in Cloud Computing environments. In order to evaluate how well the tools scale, both the Reverse Link Graph experiment and the PageRank experiment will be run using a varying number of nodes, with the difference in runtime evaluated to determine how well adding additional resources impacts the data processing tools.

Both Nephele and Hadoop (and in a broader sense, PACT and MapReduce) are based on stateless functions being distributed across nodes, evaluating data massively in parallel. Because of the nature of this model, communication overhead (the primary cause of parallel overhead) should be minimised during the primary processing time of the tools. It can therefore be expected that both Nephele and Hadoop will increase in a generally linear fashion.

It would be unrealistic to expect the tools to scale in a perfectly linear manner, as there is still communication required between the nodes. In particular, some operations can be made parallel more easily than others; map is a naturally parallel action, where the nature of reduce requires communication between at least 2 completed map operations. There will also be communication required at the start and end of each job, as input and output data must be read from, and written to, the file system. Finally, there will be overhead required to schedule tasks to nodes throughout the cluster. This process is centralised, with the master node determining which nodes will be responsible for processing different parts of the data at each stage. This computation is serial by necessity, and in contrast to most other processing tasks may increase in time as more nodes are added (as the scheduling problem will become progressively more complex).

One major cause of scalability issues in parallel computing can be the underlying file system. Communicating to disk is expensive, and will be even more so if a node has to communicate over the network to get access to the file system. The file system implementation may therefore play a key role in determining the scalability of the system. An inferior file system implementation may lead to more network communication, significantly negatively impacting the runtime of a processing task. As both Nephele and Hadoop use the Hadoop Distributed File System, this discrepancy will not be an issue when assessing the relative scalability of the two systems.

Based on this analysis, it can be expected that Nephele and Hadoop will not show any major differences in scalability, with both scaling in a near-linear fashion.

Chapter 4

Experiment Implementation

This chapter will cover the implementation details of the experiments, discussing the tools used, the deployment environment and other considerations made when running the data processing tasks.

4.1 Choice of Data

An important factor in big data problems is the nature of the data which is being processed. As discussed in the Background Reading (chapter 2) data is usually large scale, but may also be unpredictable, changing in both size and quality. It is important to choose a dataset representative of what data processing tools are usually used for, as using completely synthetic data without appropriate properties will not provide an accurate portrayal of how well the tools perform in the real world.

4.1.1 Desirable Properties

There are several desirable properties which will make a dataset appropriate for use in the experiments.

4.1.1.1 Connectivity

Both the Reverse Link Graph and PageRank problems treat the input data as graphs, and perform operations on them accordingly. An important property that the data must have is that the graph must have a high degree of connectivity (i.e. lots of connections between nodes in the graph).

A Reverse Link Graph relies on a node being linked to by other nodes in the graph. In the case where there are no links between nodes, the algorithm will not generate any link pairs, and will therefore have no work to perform in the Reduce stage. The more links pairs are generated, the greater the amount of work required in the Reduce step.

In the case of PageRank, the algorithm uses values of adjacent nodes to determine a nodes value. A node which is connected to many high valued nodes will have a high value itself. This requires high degree of connectivity to provide a meaningful PageRank. It also requires that all nodes be in the graph.

The subset of a graph cannot be used as it will have ‘dangling references’ - connections where one of the nodes is outside of the graph (and therefore has no PageRank).

4.1.1.2 Size

Size is an important property of the data as it should be reflective of real-world data sizes. In particular, the size of the data is constrained by 2 major factors.

1. The size of the data cannot be too *small* as it will limit the tools capacity to run the problem in parallel. Amdahl’s law [8] states that there is a theoretical maximum speedup which can be achieved by adding additional parallel processing capacity. Past this point, adding extra resources (nodes or processors) will not result in improved runtime performance, and may detract from performance (due to additional communications overhead, or other serial factors).

2. The size of the data cannot be too *large* as there will be issues with storing, transferring, and processing the data on the resource constrained deployment environment.

4.1.1.3 Ease of Processing

There are various different areas where Reverse Link Graph and PageRank can be used. Examples include analysing the importance of academic papers, or determining which sections of code in a program are dependant on one another. One of the major factors in choosing which type of data is used is how well suited it is to be processed by the data processing tools.

Specifically, whilst it is possible to process other types of data, it is much simpler to process plain-text data, as there are a wide array of available input mechanisms. Both the Hadoop and Nephele APIs have methods of reading in a text file, processing CSVs and handling plain text in other ways.

Many academic papers are made available in PDF format. This makes it difficult to process as custom input adaptors would have to be created to parse the data and pass it to the data processing steps. This would have limited impact on the performance of the tools, and is therefore not important within the scope of the project.

An ideal candidate for a naturally occurring, plain text representation of a graph would be the World Wide Web. The web consists of a set of pages (nodes) with links (edges) between them. Treating the web as a graph is a common feature of other research works, including the original PageRank implementation [31].

4.1.2 Wikipedia Datasets

Wikipedia is an online encyclopaedia which is available in multiple languages. It exhibits all of the properties which are desirable for the data processing problems. Wikipedia represents a graph with high connectivity, as articles within the encyclopaedia often link to other articles. The Wikimedia foundation make a set of full HTML dumps available for the various language editions of Wikipedia dating up to 2008. As a plain text representation of the Wikipedia dataset, this is easier to process than alternative formats (such as PDFs).

The various language versions of Wikipedia are different sizes. This is allows prototyping to be carried out with different data sizes, in order to find the appropriate size for the experiments.

- Wiki for Schools is a charitable project aimed at providing a self-contained subset of Wikipedia for use in schools. Pages are chosen for the subset of the encyclopaedia based on whether they are part of the English school curriculum. The Wiki for Schools dataset was evaluated, but was deemed too small to provide a true test of the capabilities of the data processing tools.
- The Spanish language version of Wikipedia was also evaluated, but was too large to be used for data processing, as the resource constraints prevented experiments from being run on a low number of nodes.
- The Simple English Wikipedia is an encyclopaedia for explaining complex subjects in layman's terms, without using jargon or assuming any prior knowledge of a subject. The simple English Wikipedia provided a good compromise in data size, and was therefore chosen as the dataset for running the experiments.

4.2 Data Format

In addition to the representation of the data (a plain text representation) the underlying format used to store this data is also important. The Hadoop File System is known to suffer from the "Small File Problem" [42]. The file system is designed and optimised to store large files, rather than accessing lots of smaller files. In addition to this, HDFS has a large block size. Each file takes up at least one block which is by default 1MB. This will cause a significant amount of space to be wasted if each file is stored independently (as web pages are rarely 1MB in size).

The choice was taken to pre-process the data used in the experiments in order to overcome the small file problem. Whilst this represents an inherent limitation of the technology used, it effects both Stratosphere and Hadoop equally (as both use HDFS) and therefore will not have a dramatic effect on the conclusions drawn from the experiments. It will also allow both data processing technologies a chance to perform in ideal circumstances, rather than limiting them by the choice of data. This may not reflect a real-world implementation of the processing tasks, as this pre-processing step may be prohibitively expensive in larger datasets (for example, if the whole World Wide Web was being considered).

The pre-processing step combines the small files together into a larger file, which can then be processed accordingly.

4.2.1 SequenceFile

A SequenceFile is a Hadoop-specific binary file format which stores Key/Value pairs. It is optimised for use in HDFS. A tool was created which converted an arbitrary set of web pages into a SequenceFile, by setting the key as the filename of the web page, and the value to the contents of the file. This dramatically reduced the size of the data, as it discarded irrelevant parts of the data (such as image files) and provided a more compressed representation of the text data.

Unfortunately, the Stratosphere implementation of the SequenceFile input format was felt to be too immature for use in the experiments, and therefore the use of a SequenceFile to represent data was deemed unsuitable.

4.2.2 CSV

CSV (Comma-separated values) is a plain-text method of representing tabular data. CSV is a well established method of storing data, and while it lacks an official standard tools which process CSV data typically follow RFC 4180 [34]. As an open and accepted data format, CSV is well supported by both Hadoop and Stratosphere. The relative simplicity of the format also makes it feasible to implement a basic parser where necessary.

Data was stored in CSV format as Key/Value pairs, with the key being the page filename, and the value being the contents of the file. This mirrors the way the data was stored in a SequenceFile, but in a more widely accepted file format.

4.3 Deployment Environment

All experiments were run using the Leeds University School of Computing Cloud Testbed. The Cloud Testbed is a private cloud consisting of 6 machines, controlled using the OpenNebula toolkit. Virtual machines can be created as required, and this capability was used to add resources to the data processing tools.

4.3.1 Node Specifications

The Cloud Testbed allows the creation of Virtual Machines with fixed resources. The virtual machines are each permitted 1 virtual processor core and 1GB of memory. This allows the virtual machines to have a consistent level of performance, limiting the effect of load on the underlying hardware. The physical hardware the virtual machines execute on have Intel Xeon Quad Core processors (X3360) and 4GB of memory.

The Cloud Testbed allows for up to 10 virtual machines to be created per user. Whilst data processing software such as Hadoop has been designed to use large numbers of nodes, and this cluster size may be smaller than in some real-world applications of Hadoop, significantly smaller cluster sizes are used in practice. For example, Amazon limit users of their MapReduce service to 20 nodes, and require anybody exceeding that usage to request a higher quota [5]. This approach is consistent with previous research in the field [40].

The resources provided by the Cloud Testbed are not ideal hardware for running large scale data processing problems, which typically require large amounts of memory. Whilst this will undoubtedly impact the performance of the experiments, the environment is consistent for both Hadoop and Stratosphere, and shouldn't affect the comparison of the tools.

4.3.2 Software

Hadoop version 2.3.0 and Stratosphere version 0.5 were installed on the cluster. At present these are the latest versions of both software packages. Stratosphere was configured to use HDFS in order to limit the effect that the file system had on the experiments. The version of Stratosphere installed was not dependant on YARN, as it was felt this would limit the effectiveness of testing scalability.

The data was loaded into HDFS before the experiments, and this was not included in runtime calculation. This limits the cost of communication overhead, as the data is already distributed across nodes appropriately, rather than data having to be transferred at the start of experiments.

4.4 Programming Language & Frameworks

Experiments were initially implemented in the Scala programming language [1]. Scala is a functional programming language with Object-Orientated support, which uses the Java Virtual Machine as a runtime environment. This allows seamless integration with existing Java libraries, such as those exposed by Stratosphere and Hadoop. As a functional language, Scala has native support for higher order functions such as map - indeed, MapReduce was originally inspired by some of the fundamental operations present in functional languages [18]. This allows PACT and MapReduce jobs to be represented in a very succinct and readable manner.

In addition to being able to call existing Java libraries, there are several 'wrapper' libraries which have been developed to allow for Hadoop jobs to be written in idiomatic Scala, taking full advantage of functional features such as higher order functions. Prototypes were creating using Scalding [37], a library developed by Twitter, but use of this framework was later discontinued as it lacked support for later versions of Hadoop. It was replaced by Scoobi [29], an open source project by NICTA.

The standard distribution of Stratosphere features a Scala API, but it was found to be too unstable for experiment development, with significant changes to the interface between different software versions. For this reason, Stratosphere experiments were reimplemented in Java, whilst reusing the core, technology agnostic processing logic which was created in Scala.

The difference between Java and Scala is not significant in regards to processing performance. Both languages compile to Java Virtual Machine bytecode, and are therefore largely interchangeable.

4.5 Reverse Link Graph

The Reverse Link Graph can be computed using a single Map and Reduce operation. The implementation is therefore very similar across Hadoop and Stratosphere, as it does not use any of the additional functionality present in PACT.

The Reverse Link Graph experiment was run a number of times to give an average runtime performance, as runtime can be affected by a number of external elements.

The first step of computing the Reverse Link Graph is the Map operation. The Map stage operates on each of the documents (or articles) in the Wikipedia dataset. It extracts any HTML anchor tags from the source code of the document, and returns a list of link pairs, with the first item being the destination of the link, and the second item being the source (the current document) that the link is from.

The reduce step relies on the runtime (either Hadoop or Stratosphere) to group the items by key, so the links are grouped by the destination. The reduce step can then combine the sources into a string, giving a list of the pages which link to that destination.

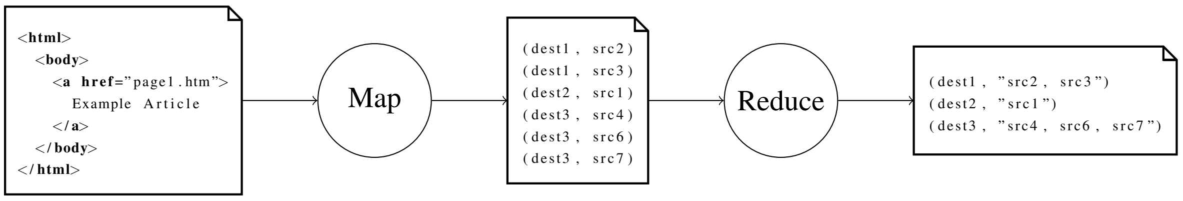

Figure 4.1: A graphical representation of the Reverse Link Graph workflow

4.6 PageRank

PageRank is a more complex algorithm than the Reverse Link Graph computation, and benefits from the more advanced features of PACT. In particular as an iterative algorithm, the PageRank algorithm benefits from having that functionality built-in to Stratosphere.

As Hadoop lacks some of the necessary operations to implement PageRank in an ideal way, the algorithm was implemented in twice, with the different technologies having different implementation styles. For Stratosphere, all of the features available to PACT were used, whereas with Hadoop the implementation is limited to being expressed in a MapReduce fashion.

The PageRank experiment was run a number of times to give an average runtime performance, as runtime can be affected by a number of external elements.

For both implementations the PageRank algorithm ran for 3 iterations. This was in order to provide a fair illustration of how well iterative algorithms perform in both technologies, whilst not being prohibitively time consuming (especially on smaller cluster sizes).

4.6.1 Stratosphere Implementation

The stratosphere implementation of PageRank was the 'ideal' implementation, as it has a number of operations required by the PageRank algorithm.

The first step in the process was to take the input data from the data source, and build the Reverse Link Graph. The PageRank algorithm determines the rank of a page by considering other pages which link to it, which requires the Reverse Link Graph to be computed.

Once the Reverse Link Graph had been found, the iterative process could begin.

Each node received the PageRank of its neighbours using a Join operation, finding the scores of other nodes by their unique identifiers (the page name). The resulting data was then sent to a Reduce operation, which computed the new rank of the node by summing the rank of its neighbours, and multiplying the resulting value by a 'dampening value'.

After the first map operation, this process was applied iteratively 3 times.

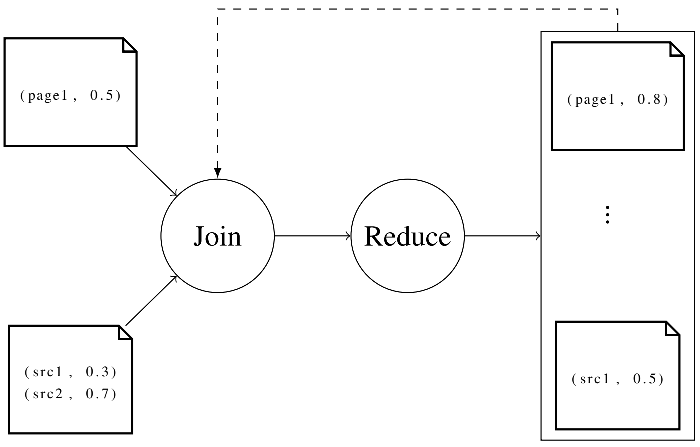

Figure 4.2: A graphical representation of the iterative part of PageRank

4.6.2 Hadoop Implementation

As Hadoop lacks the necessary operations to perform the PageRank algorithm, it must be implemented using a series of Map and Reduce operations.

The initial process of converting the input data to the necessary format by performing the Reverse Link Graph calculation took a single Map and a Reduce operation.

For each iteration of the algorithm, a Map and Reduce operation was required to join the node with its edges. A further MapReduce step was required to compute the updated rank for the page (as with Stratosphere). A final MapReduce step was taken to format the data in an appropriate format for the next iteration, a step taken automatically by Stratosphere.

It took a total of 10 MapReduce passes to compute the PageRank over 3 iterations.

Chapter 5

Technical Evaluation

As discussed in the Experiment Design and Implementation sections, each problem was run on both Hadoop and Stratosphere with a varying number of nodes. The same data was used for all of the experiments, and a runtime average was taken to provide the best possible indication of each tools performance.

This chapter provides a technical analysis of the results gained from experimentation, and relates them back to the hypothesis.

5.1 Reverse Link Graph

5.1.1 Hadoop

The Hadoop implementation of the Reverse Link Graph is important as a benchmark against which other experiments can be judged. The performance of Hadoop in MapReduce situations is a well-known quantity, as it is the most widely used data processing technology. This provides a clear metric against which the performance of Stratosphere can be evaluated, whilst also serving to illustrate how well Hadoop performs MapReduce tasks in comparison to non-MapReduce tasks.

Number of Machines	Average Runtime	Standard Deviation
2	430.60	144.20
4	274.60	41.72
6	211.20	31.91
8	172.40	15.70
10	173.40	3.28

Figure 5.1: Hadoop Reverse Link Graph Raw Data

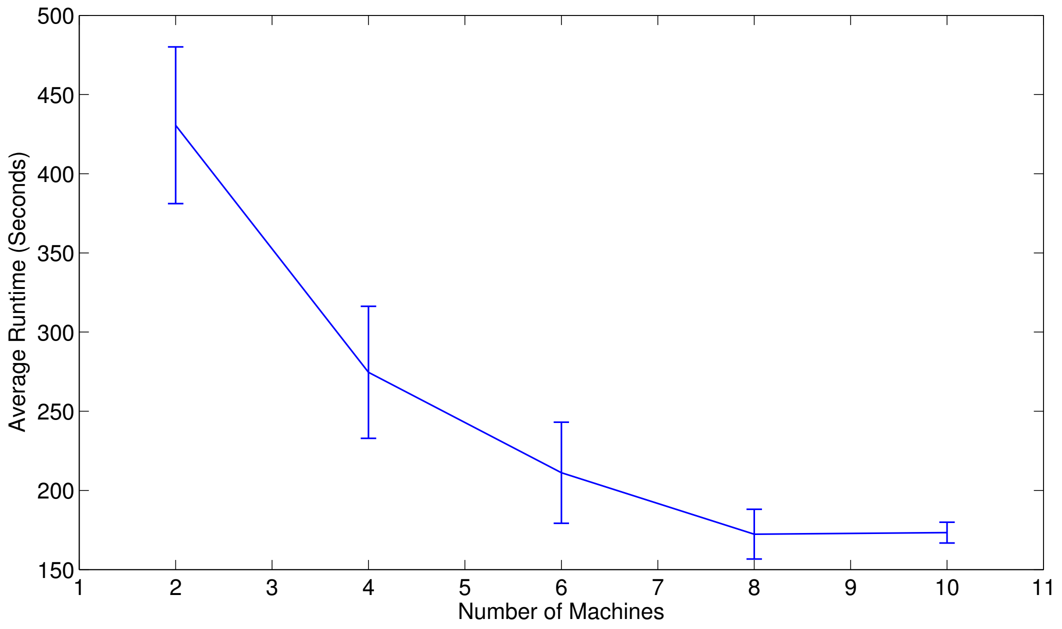

Figure 5.2: Hadoop Reverse Link Graph Runtime

The Hadoop implementation of the Reverse Link Graph problem shows the general properties expected from the data. As the number of nodes increases, the runtime decreases as having a larger amount of resources allows more work to be processed in parallel. This trend reverses when the experiment is executed on 10 machines. This typically indicates that a point has been reached where a problem no longer benefits from adding additional parallel resources, and it becomes more costly to run on a larger number of machines because of communications overhead. The impact of Amdahl's law [8] is not indicative of a problem with Hadoop, and is likely caused by the size of the data, as discussed in the Experiment Implementation section.

The expected trend of performing scaling near-linearly as the number of machines increases can be seen more clearly in figure 5.3. This exhibits the expected behaviour that the runtime nearly halves as the number of machines is doubled. As expected, the runtime is not quite linear as there is some communications overhead added by adding more machines to the cluster. This reflects the change that can be seen in figure 5.2 toward the end of the plot. Figure 5.3 does not show the more dramatic decrease in performance that can be seen when the experiment is run on 10 machines, as it only features the runtime for 2, 4 and 8 nodes. This effect would be shown if the experiment was performed on 16 nodes.

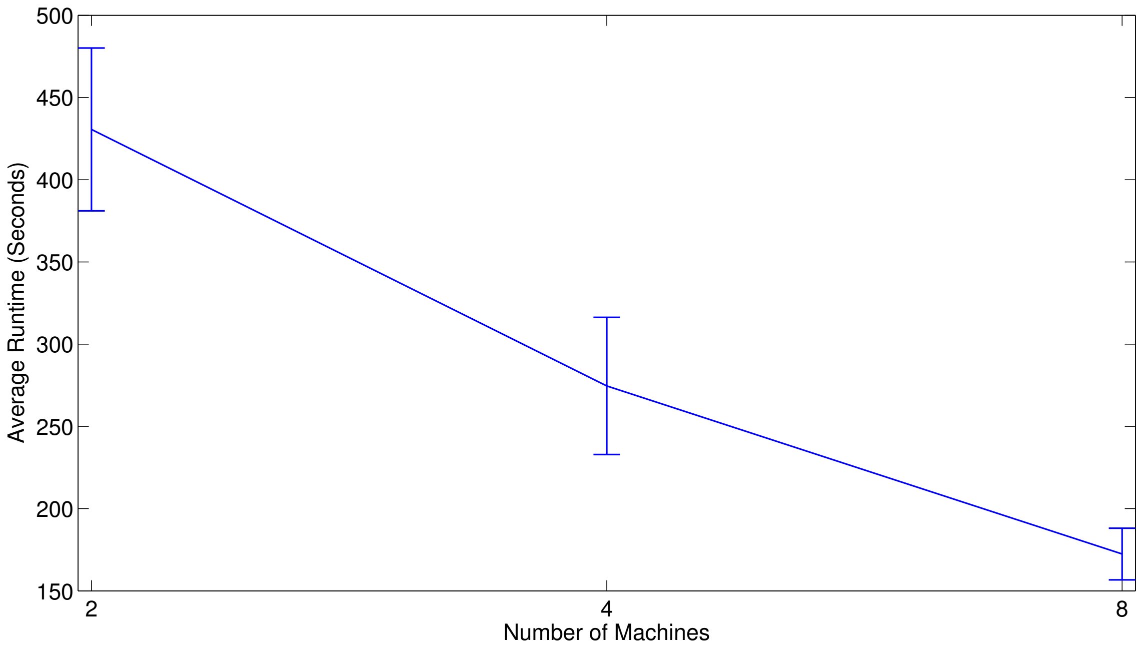

Figure 5.3: Logarithmic plot of Hadoop Reverse Link Graph Runtime

Another important factor of figure 5.2 is the error bars, indicating the standard deviation of runtime. The standard deviation is initially fairly high, as the experiments take a (relatively) long time to run. As the number of machines increases, there becomes less room for error as the time taken is decreased.

5.1.2 Stratosphere

The Stratosphere implementation of the Reverse Link Graph experiment allows a comparison of how well it can perform standard MapReduce tasks in comparison to Hadoop.

Number of Machines	Average Runtime	Standard Deviation
2	144.20	1.92
4	92.20	2.04
6	72.00	1.87
8	62.20	3.63
10	59.40	3.28

Figure 5.4: Stratosphere Reverse Link Graph Raw Data

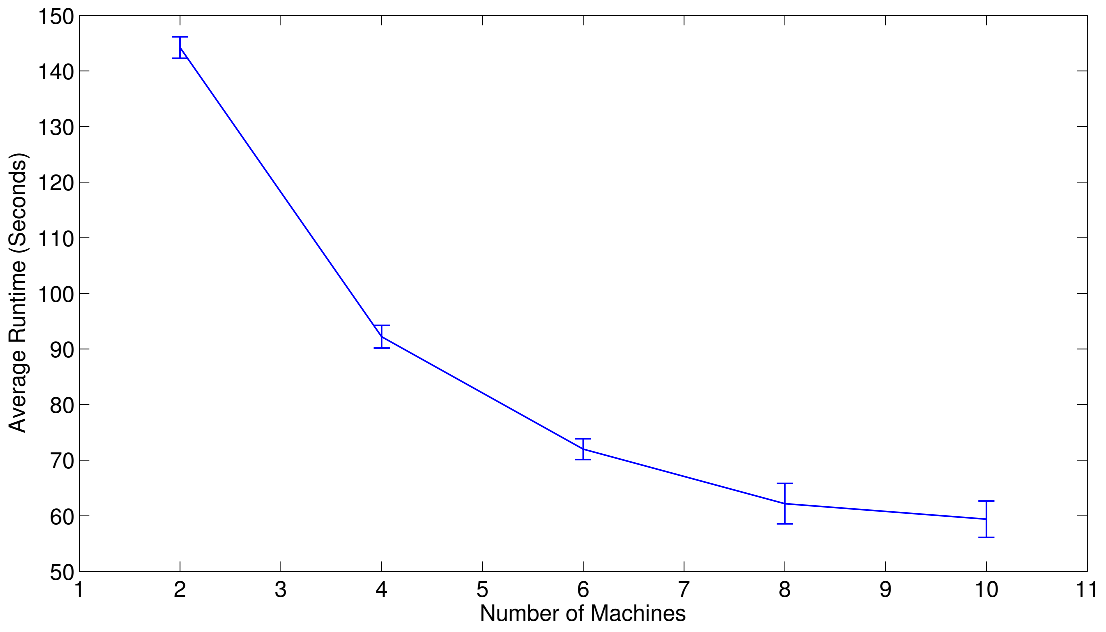

Figure 5.5: Stratosphere Reverse Link Graph Runtime

The Stratosphere implementation also shows the expected speed increase as extra nodes are added to the cluster. As with the Hadoop implementation, it begins to suffer from Amdahl's law towards the higher number of nodes in the cluster. Unlike with the Hadoop implementation, running the experiment on 10 machines does not increase the runtime, but the performance enhancement from adding more nodes clearly diminishes as the cluster gets larger.

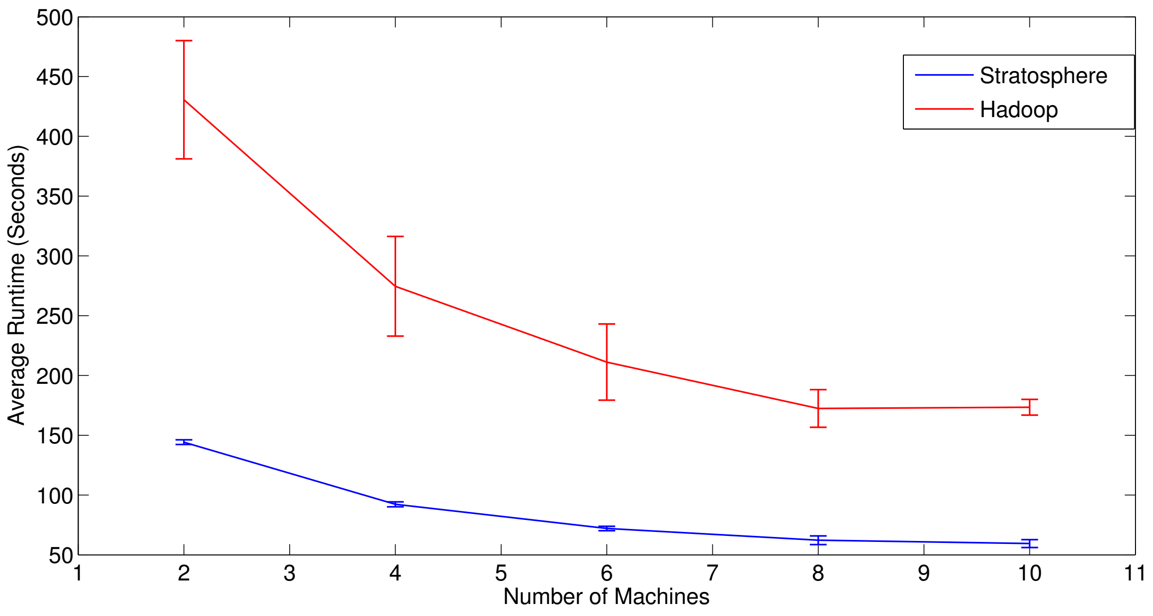

Figure 5.6: Comparison of Hadoop and Stratosphere Reverse Link Graph Runtime

When comparing runtime of the Hadoop and Stratosphere Reverse Link Graph implementations, it is clear that the Stratosphere experiment runs much faster across all cluster sizes. This contradicts the hypothesis, which expected that Hadoop would perform better for classic MapReduce problems. Potential reasons for this are explored in section 5.3.

In addition performing better, it appears that Stratosphere has a more predictable runtime. The standard deviation of Stratosphere experiments is much smaller, indicating that there is less variance in runtime. This may indicate that Stratosphere is less susceptible to the varying factors of the environment it is executing in, such as network congestion or disk usage.

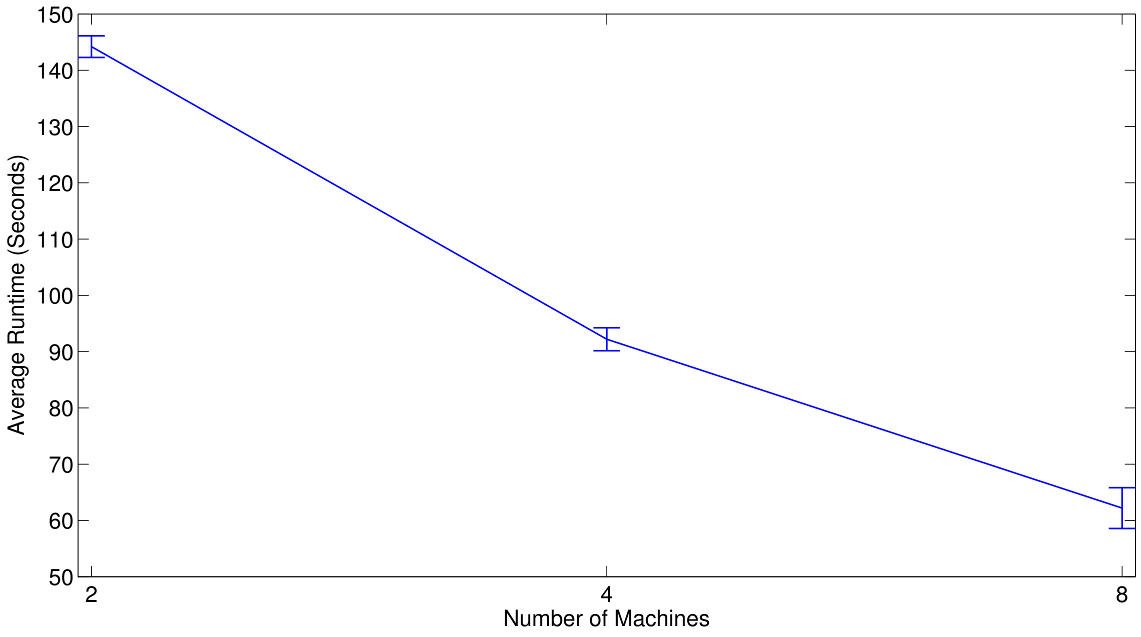

Figure 5.7: Logarithmic plot of Stratosphere Reverse Link Graph Runtime

The Stratosphere implementation of Reverse Link Graph also scales near linearly to the amount of processors. When comparing to the Hadoop implementation, it appears that the implementation may scale slightly worse between 4 and 8 nodes, though this is likely to be because of the effects of Amdahl's law. Stratosphere would be more susceptible to the issues of parallel overhead as its runtime is lower, and communication would therefore take up a higher percentage of the overall execution time.

5.2 PageRank

5.2.1 Hadoop

The Hadoop implementation of PageRank serves to show that the MapReduce paradigm is not capable of expressing all algorithms in an efficient manner. It clearly demonstrates a potent disadvantage of the MapReduce paradigm, and provides a point of comparison for the equivalent Stratosphere implementation.

Unfortunately, due to resource constraints on the cluster, it was not possible to compute the solution to PageRank using Hadoop on 2 or 4 nodes. This shows the ineffective nature of Hadoop when attempting to solve iterative problems. It also prevents an assessment of the scalability of the solution, as it was not possible to perform the task on half the number of nodes.

Number of Machines	Average Runtime	Standard Deviation
6	3429	150.58
8	3197	132.86
10	3060	117.06

Figure 5.8: Hadoop PageRank Raw Data

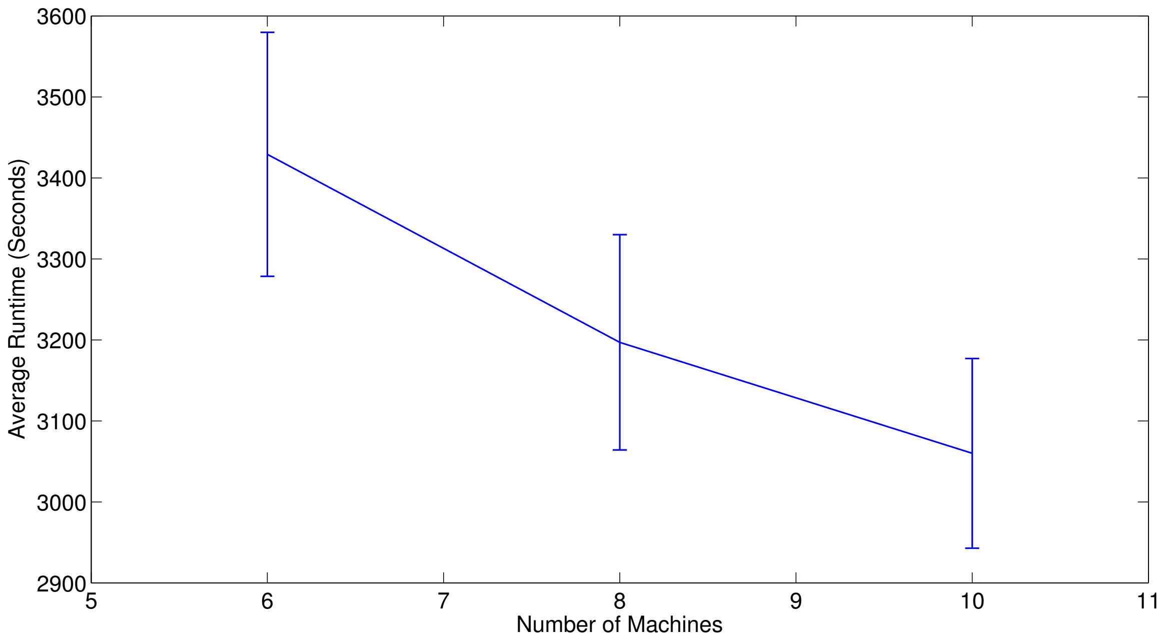

Figure 5.9: Hadoop PageRank Runtime

Despite using the same dataset, PageRank is much less efficient to compute in Hadoop than the Reverse Link Graph is. This can be seen clearly in figure 5.10. Whilst it can be expected that PageRank will take longer to compute (with the Reverse Link Graph being a component of PageRank) the size of the difference shows that Hadoop is sub-optimal when performing iterative algorithms. A large part of this is that each iteration is implemented as a number of Map and Reduce steps. There is overhead for every MapReduce job submitted to Hadoop, as it must schedule the work across nodes, load the appropriate data, perform the computation and write the intermediate data to disk before moving on to the next step in the process. The overhead of this process would be less significant where larger data sizes are used, but would increase as more iterations were used (requiring more Map and Reduce steps), and as the cluster size increases (increasing the associated start up time).

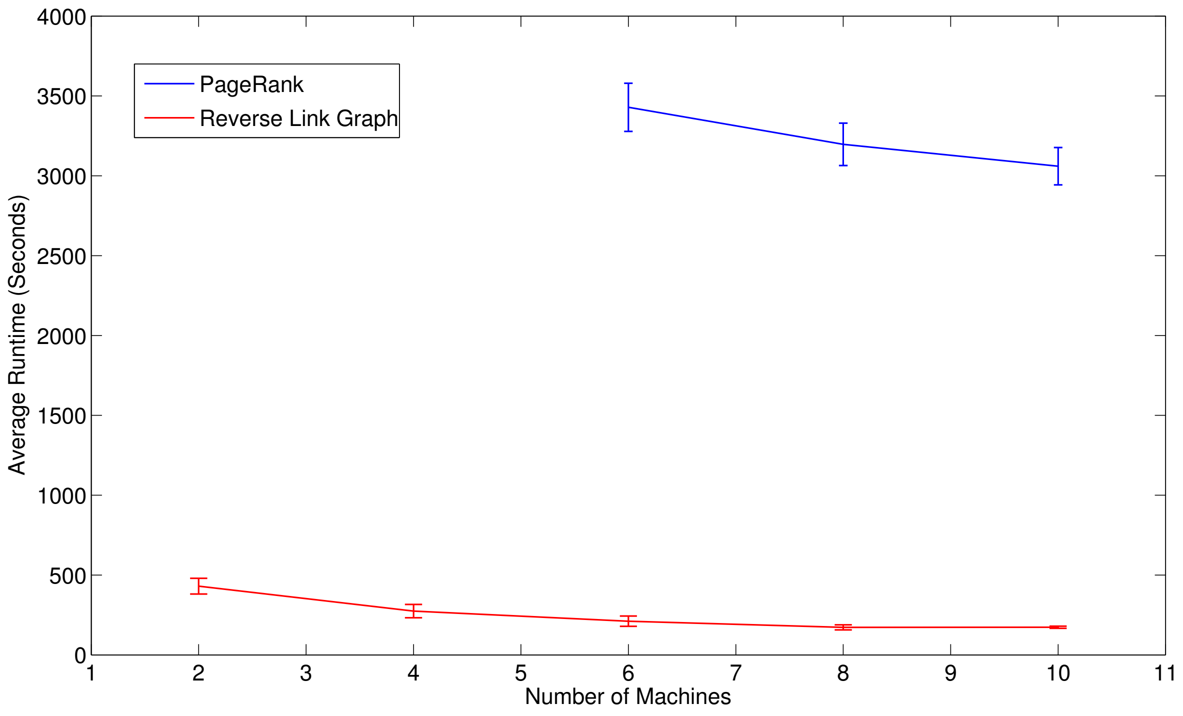

Figure 5.10: Hadoop PageRank and Reverse Link Graph Runtime Comparison

5.2.2 Stratosphere

The Stratosphere implementation of the PageRank algorithm shows the performance increases that are possible when using a more generalised paradigm such as PACT over MapReduce.

Number of Machines	Average Runtime	Standard Deviation
2	588.00	0.57
4	565.66	2.51
6	553	2.64
8	533.66	2.08
10	522.00	2.64

Figure 5.11: Hadoop PageRank Raw Data

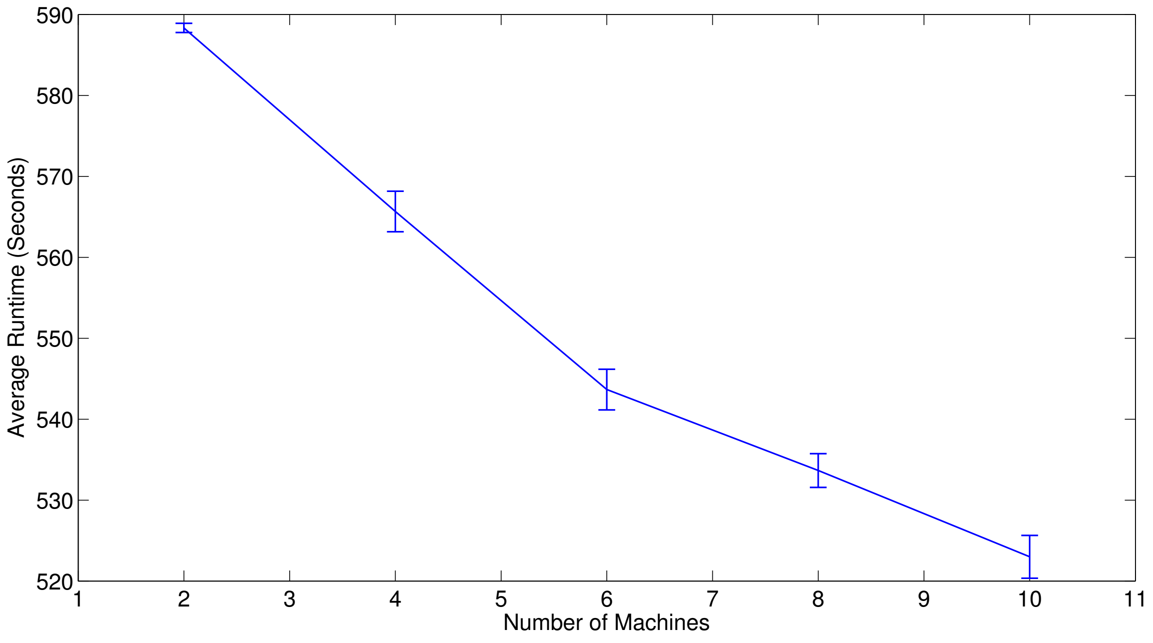

Figure 5.12: Stratosphere PageRank Runtime

The Stratosphere runtime performs significantly better than Hadoop when performing PageRank. In addition from benefiting from superior runtime in simple MapReduce cases (as show by the earlier experiments), Stratosphere benefits from having built-in support for iterations, and the ability to create complex Directed Acyclic Graphs representing the operations which must be carried out. This benefits Stratosphere as intermediate results can be stored in memory, and data does not necessarily have to be shuffled before subsequent processing steps can begin. This reduction in I/O and network overhead play a large role in improving the runtime performance of the experiment.

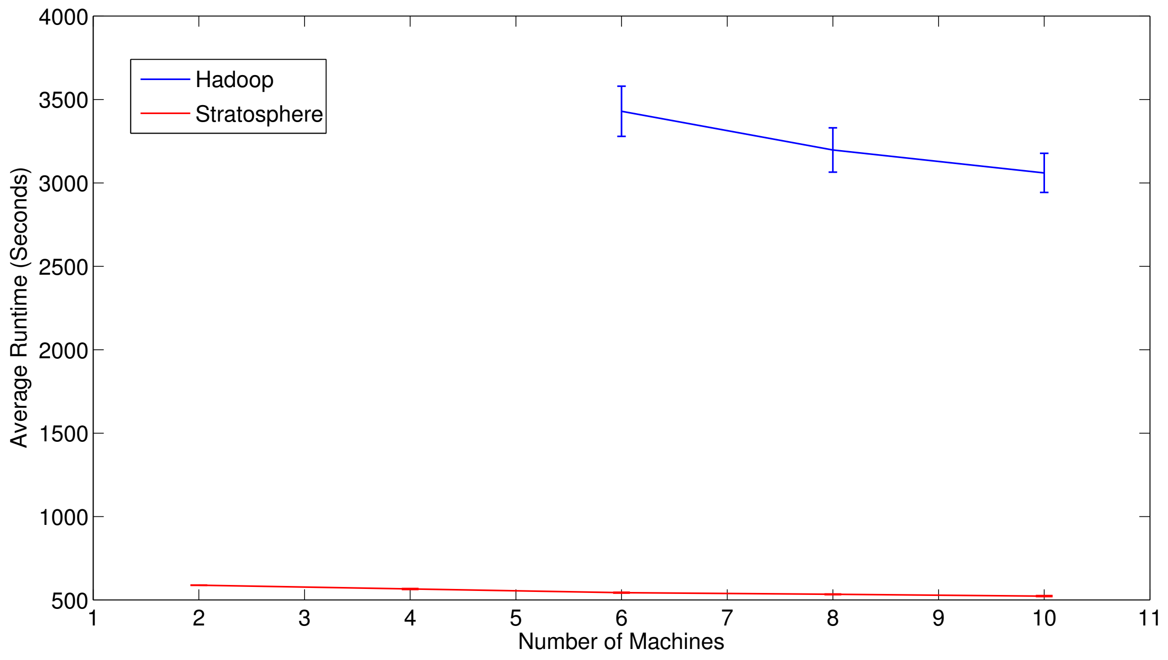

Figure 5.13: Comparison of Hadoop and Stratosphere PageRank Runtime

Unlike previous experiments, the Stratosphere PageRank implementation appears to handle the additional 2 machines from 4 to 6 in a similar manner to the improvement from 2 to 4 nodes. This is due to the poor speedup observed when extra nodes were added to the cluster. This behaviour can also be seen when adding machines to the Hadoop cluster, and we can therefore surmise that this is due to the nature of the problem, rather than an inherent behaviour of the runtime.

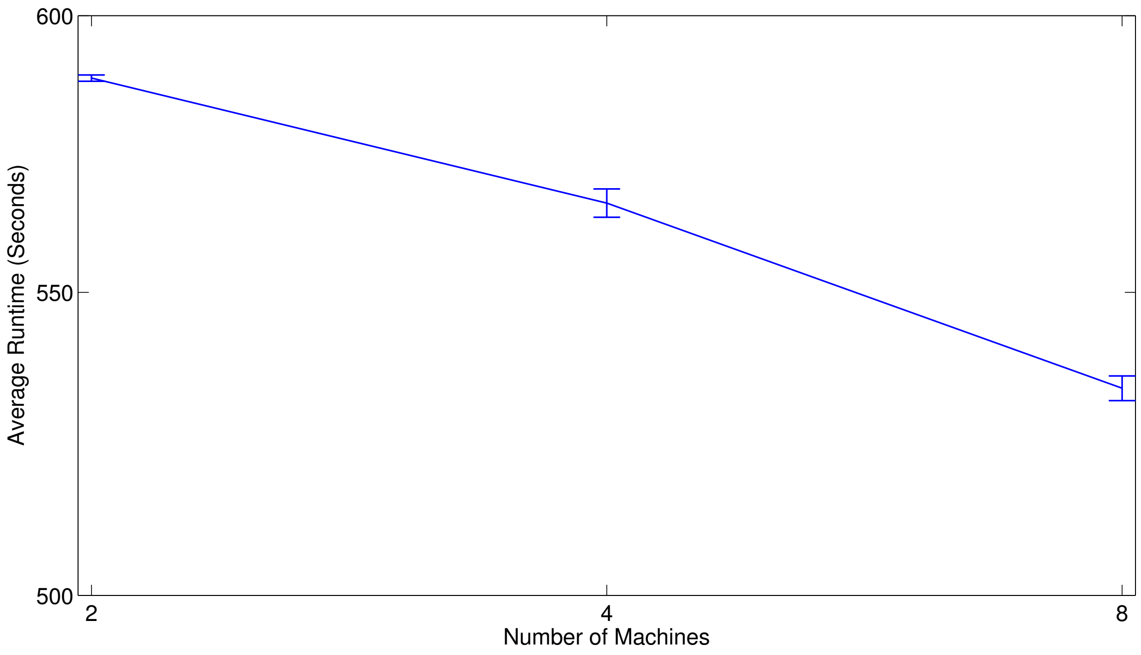

Figure 5.14: Logarithmic Plot of Stratosphere PageRank Runtime

5.3 Hypothesis Evaluation

The hypothesis formed in the Experiment Design consisted of 3 core statements which summarised the expected outcome of the experiments.

5.3.1 Hadoop outperforms Stratosphere in MapReduce Tasks

It was expected that Hadoop would outperform Stratosphere when the data processing job was one which could inherently be expressed in a MapReduce manner. This part of the hypothesis was disproved, as Stratosphere performed consistently better when computing the Reverse Link Graph. After further analysis, there are various reasons why Stratosphere may perform better.

As indicated in previous research [40][11] Stratosphere writes to disk less than Hadoop, with disk I/O typically being a bottleneck in this type of task. Stratosphere is also more likely to utilise nodes fully, using upwards of 80% of each machines CPU. Hadoop is more likely to allow system resources to be idle.

Perhaps the largest reason behind the difference in runtime is that Stratosphere is capable of running each step of the computation in parallel. Whilst Hadoop will begin copying the data from Mappers as they complete, the actual Reduce logic will not be executed until all Mappers have finished execution [41]. In contrast, Stratosphere will begin the Reduce step as soon as there is data and computing resources available.

5.3.2 PACT improves upon MapReduce

It was hypothesised that PACT would improve upon the core shortcomings of the MapReduce paradigm, by performing better at iterative algorithms.

The results of the Hadoop PageRank experiment clearly indicate that MapReduce is performs significantly worse at iterative algorithms than with other problems, due to the poor runtime in comparison with the Hadoop Reverse Link Graph experiment (despite using the same data). This indicates a problem with the broader MapReduce paradigm.

Stratosphere performs significantly better than Hadoop at computing the PageRank. This is in a large part due to PACT; it is greatly supported by having support for iterations without having to write to disk, and without having to submit several processing jobs to complete one logical task. Having a more expressive range of operations is also clearly to PACTs advantage. In the PageRank experiment, Stratosphere benefited from having a dedicated Join operation rather than having to use a Map and a Reduce.

5.3.3 Hadoop and Stratosphere scale near-linearly

The final statement in the hypothesis was that Hadoop and Stratosphere would scale in similar terms, both near-linearly. When considering scalability, an important metric is parallel speedup, which can be defined as the following formula:

$$S_p = \frac{T_1}{T_p}$$

In the case of tools such as Hadoop and Stratosphere, it is difficult to get a 'serial time', as it would be artificial to run experiments on one node. This would be unreliable as the technologies are designed for use in clusters of machines, and there is therefore no clear definition of 'serial runtime'. However, we can use the concept of speedup to measure how well the data processing tools scale, by simply comparing the average runtime with that of experiment run on half the number of nodes. For example

$$S_4 = \frac{T_2}{T_4}$$

Should the data scale perfectly (i.e. linearly), this value would always be 2, as runtime would half as the processor count doubles.

When this is calculated for the Reverse Link Graph experiment it yields the following results. As there is not a complete set of results for the PageRank experiment, it will not be considered.

Number of Processors	Hadoop Speedup	Stratosphere Speedup
4	1.568	1.592
8	1.563	1.482

Figure 5.15: Reverse Link Graph Speedup

This indicates that the hypothesis is correct. With both Hadoop and Stratosphere the algorithms get roughly 75% faster when the processor count doubles. This is sub-linear (i.e. not 100%) owing to communication overhead, and other inherently serial factors in execution. As with all parallel algorithms, there is a point at which the serial cost of adding new nodes outweighs the benefit they give to the computation. This process is clear with Stratosphere as the number of processors increases, as the runtime is shorter causing serial elements to take up a greater proportion of the computation.

5.4 Conclusion

Several experiments have been designed and performed with the aim of determining whether Stratosphere can perform better than Hadoop, and whether the generalised execution framework that PACT provides has any real benefits of Hadoop.

The results of these experiments have shown that Stratosphere outperforms Hadoop in cases where a problem both can and cannot be clearly expressed as MapReduce problem, with PACT playing a key role in enabling these performance improvements.

Whilst this indicates that Stratosphere is a more efficient tool in the general case for large scale data processing, there are several other considerations that should be made when choosing a tool for real-world applications.

Chapter 6

Project Evaluation

This chapter will evaluate the Final Year Project as a whole, focusing on the process, methodology and schedule of the project, and discussing its relevance to the larger research landscape.

6.1 Methodology Evaluation

Applying an Agile methodology was a key factor in successfully completing the project. The original project plan was very ambitious, and it quickly became clear that it was infeasible to complete this much work within the limited time frame of the Final Year Project. Following an Agile process was essential in prioritising the most important work, ensuring that a minimum viable amount of work was completed as quickly as possible, empirically adjusting the schedule as required throughout the project and forming contingency plans where necessary.

The methodology can be deemed successful as all of the minimum requirements were met. Minimum requirements were prioritising and carried out in earlier sprints to ensure that sufficient amount of work was completed for the project. When the higher priority minimum requirements were met, the additional goals were prioritised based on their impact to the project and carried out in priority order.

Agile development provided a framework for adjusting the schedule throughout the project. Reviewing progress every week, and using this information to inform the next weeks estimates made the estimation process much more accurate. Previous weeks provided evidence of how long to realistically expect a task to take, helping to avoid continually overestimating the work that could be completed.

6.1.1 Schedule Adjustments

Various adjustments were made to the schedule to ensure that the project was completed on time. Initially, the project was fairly ambitious in scope. Unfortunately, because of time constraints not all of the extended requirements could be met. Carrying work out in an iterative manner was key to the ability to remove work from the schedule (thereby ensuring the project could complete on time) whilst ensuring the minimum requirements were still met.

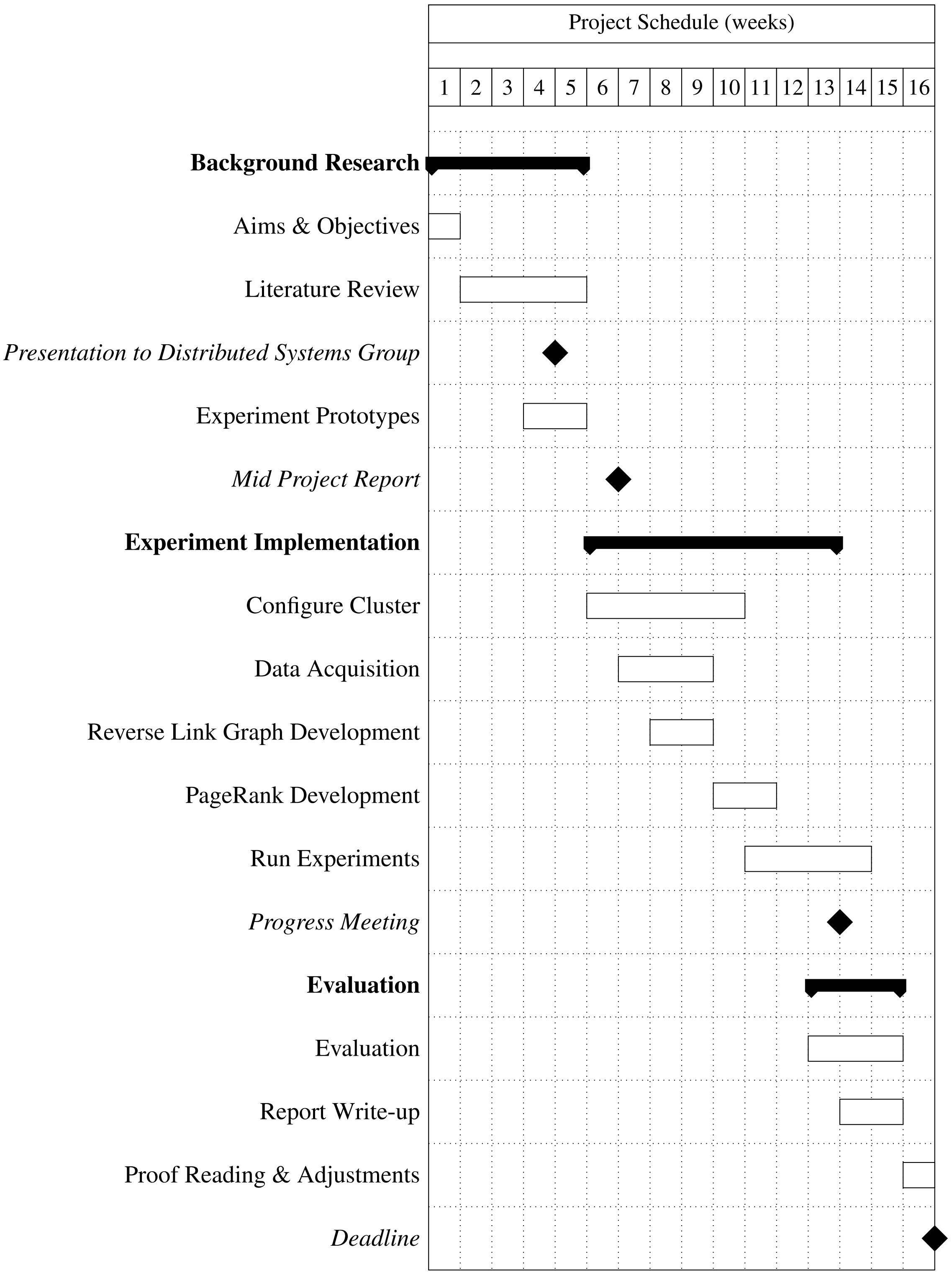

Figure 6.1: Final Gantt Chart of Project Schedule

In comparison to the initial project estimation, setting up the cluster of machines took much longer than expected. It was expected that Hadoop and Stratosphere would be pre-installed on the Cloud

Testbed, but this was not the case for Stratosphere, and Hadoop had an outdated version. The process of installing and configuring the software took longer than expected, leading to the cluster set-up to take 4 weeks, rather than the week originally predicted.

In order to compensate for this, the data required for the experiments was acquired and pre-processed as the cluster was being configured. As stages in these tasks were passive (e.g. waiting for large amounts of data to download) these tasks could be performed in parallel efficiently.

Rather than performing the experiments required for the minimum requirements (e.g. Word Count or Terasort), more complex experiments were developed to better reflect the real-world usage of the data processing tools. This met an additional requirement (in addition to the minimum requirements). Implementing the more complex experiments directly and not performing the basic experiments was a more efficient use of time, with more value being gained from the advanced experiments, and time not being spent on performing the simpler experiments.

Experiments could be run in parallel to the evaluation and write-up stages, as experiments (particularly Hadoop PageRank) often took a large amount of time to finish without requiring any supervision. This left time to write up various aspects of the report, and evaluate experiments which had already been completed (for example, evaluating the results of the Reverse Link Graph experiments as PageRank was running).

The project was completed slightly ahead of schedule, allowing for a week of proof reading and adjustments to the report.

6.2 Achievement of Aims & Objectives

The stated aim of the project was to provide a reasoned and objective analysis of the efficiency of Hadoop and Stratosphere. This aim was met through meeting the following requirements.

6.2.1 Minimum Requirements

- Install Hadoop

  Hadoop was installed on the Cloud Testbed.

- Install Stratosphere

  Stratosphere was installed on the Cloud Testbed.

- Run a MapReduce processing task

  In the original requirements, example exercises such as Word Counting and Terassort were discussed as potential MapReduce problems which could be used to benchmark Hadoop and Stratosphere. This requirement was met by running the Reverse Link Graph experiment on both tools. This will be discussed further in the ‘Real-world problem’ extended requirement.

- Run a non-MapReduce processing task

  In the original requirements, K-Means clustering was discussed as a problem which could be used to compared the ability of Stratosphere and Hadoop to run problems that do not conventionally fit in to the MapReduce paradigm. This requirement was met by calculating the PageRank of the data. PageRank is an experiment included in both Hadoop libraries, and as an example in the standard distribution of Stratosphere. The code for both of these had to be modified to meet the input format.

- Run an experiment on a different cluster size

  This requirement was met, as all experiments were executed on various different cluster sizes.

6.2.2 Extended Requirements

- Real-world problem

  In the original requirements the use of a 'real-world' problem was discussed as it would provide a less artificial way of evaluating Hadoop and Stratosphere. It was not possible to identify a suitable problem for various reasons. The availability and size of data were a limiting factor, as was finding an experiment with sufficient literature which did not require extensive domain specific knowledge. Due to the time constraints of the Final Year Project, it was deemed infeasible to complete a dedicated real-world problem.

  Rather than running the real-world problem as a separate experiment, it was decided that the MapReduce and non-MapReduce problems should be chosen in such a way that they reflected many of the properties present in real-world scenarios. The Reverse Link Graph has various real-world applications and PageRank forms a core part of search engine technology. The PageRank experiment required code to be modified to match the input data, where the Reverse Link Graph experiment was developed from scratch.

  Whilst the requirement may not have been met directly, the fundamental ideas behind the extension have been met, and the value toward the project has therefore been achieved.

- Scale all experiments

  All experiments have been run on various different cluster sizes. In addition to this, rather than simply running them on 2 different cluster sizes, each has been executed on cluster sizes of 2, 4, 6, 8 and 10 machines. This extended requirement has been met.

  The only exception to this is the PageRank algorithm on Hadoop. This was only run on 6, 8 and 10 nodes as resource constraints prevented it from working on 2 and 4 nodes.

- Public Cloud

  Unfortunately the experiments were not completed on a public cloud. It was felt that this extended requirement had the lowest priority, as it would have a limited effect on the results of the experiments and required a significant investment of time and money. Running the same experiments on a different cloud would likely replicate the results, which provides limited value in the context of the experiment. Two areas which may have benefited from using a public cloud would be that experiments could have been run on a larger scale (for example, 16 nodes to continue the scalability experiments, or using larger data) and the resource acquisition technique of Stratosphere could have been evaluated. This has been identified as an area of potential future research.

6.3 Related Work

A key part of evaluating the project is viewing the research in the context of previous work in the field.

6.3.1 Comparison to Prior Work

Specifically, there are two core Nephele papers which have previously studied the performance characteristics of Stratosphere (then Nephele) by comparing it to Hadoop ("Exploiting Dynamic Resource Allocation for Efficient Parallel Data Processing in the Cloud" [40] and "Nephele/PACTs: a programming model and execution framework for web-scale analytical processing" [11]).

The experiments carried out in this project support the performance findings of the previous work, finding that Stratosphere outperforms Hadoop in cases of both MapReduce and non-MapReduce problems, with MapReduce being particularly poor when considering iterative problems.

6.3.1.1 Contribution to Research

The work carried out in this project is intended to contribute to the existing landscape by building on the work of previous research. This project attempts to contribute to the existing body of research in the following ways

1. Performing experiments designed to test the scalability of tools, and how they react to changing resources. The existing research typically uses one fixed size cluster, with some prior research focusing on a comparison between a fixed Hadoop cluster and a Stratosphere cluster using automated resource acquisition. In this project, experiments have been run across varying cluster sizes, allowing a discussion of how well the tools respond to changing resources (in the form of cluster size).

2. The experiments have been chosen to be non-trivial in nature. This is in contrast with many existing benchmarks, which use synthetic tasks (such as Word Counting or Terasort) and data created for the sole purpose of experimentation. By comparison, we have chosen problems which attempt to reflect the real-world usage of data processing tools, with existing data sources used for experimentation.

3. By providing a performance comparison and replicating the results of prior research using later versions of Hadoop and Stratosphere. This is significant as the existing performance comparisons use earlier versions of software, and since this point the Stratosphere project in particular has changed significantly (becoming an Open Source project and being accepted on to the Google Summer of Code and Apache Incubator projects).

6.3.2 Future Work

There are a number of possible extensions to the project which could be tackled as future work.

A particular area which was identified as an extended goal of the project was to look at the performance of the data processing tools in a public cloud environment. In particular, this work could look at the impact of the unique method in which Stratosphere can acquire resources rather than processing data on a fixed cluster size, and how this approach effects performance and scalability. The cost of public clouds, and whether the choice of data processing tool can reduce cost could also be an area of research.

Various cloud providers offer hosted Platform as a Service data processing tools (typically Hadoop) which allow users to process data without worrying about manually configuring a cluster. A possible extension to this work would be to consider how this effects the performance of data processing tasks, and whether the elasticity afforded by a Platform of a Service approach is appropriate for data processing scenarios.

An alternative direction would be to look at other, more specialised data processing tools which are designed to process certain types of data (for example, large scale graph processing tools).

Bibliography

[1] The scala programming language, 2014.

[2] Stratosphere programming model, 2014.

[3] Welcome to apache hadoop!, 2014.

[4] Alexander Alexandrov, Max Heimel, Volker Markl, Dominic Battré, Fabian Hueske, Erik Nijkamp, Stephan Ewen, Odej Kao, and Daniel Warneke. Massively parallel data analysis with pacts on nephele. *Proceedings of the VLDB Endowment*, 3(1-2):1625–1628, 2010.

[5] Amazon. Elastic mapreduce troubleshooting.

[6] Amazon. Aws — amazon elastic compute cloud (ec2) - scalable cloud services, 2014.

[7] Amazon. Public data sets on aws, 2014.

[8] Gene M Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485. ACM, 1967.

[9] Alex Amies, Harm Sluimen, Qiang Guo Tong, and Guo Ning Liu. *Developing and Hosting Applications on the Cloud*. IBM Press/Pearson, 2012.

[10] Sitaram Asur and Bernardo A Huberman. Predicting the future with social media. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on*, volume 1, pages 492–499. IEEE, 2010.

[11] Dominic Battré, Stephan Ewen, Fabian Hueske, Odej Kao, Volker Markl, and Daniel Warneke. Nephele/PACTs: A programming model and execution framework for web-scale analytical processing. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pages 119–130, New York, NY, USA, 2010. ACM.

[12] Johan Bollen, Huina Mao, and Xiaojun Zeng. Twitter mood predicts the stock market. *Journal of Computational Science*, 2(1):1–8, 2011.

[13] André B Bondi. Characteristics of scalability and their impact on performance. In *Proceedings of the 2nd international workshop on Software and performance*, pages 195–203. ACM, 2000.

[14] Gerard Briscoe and Alexandros Marinos. Digital ecosystems in the clouds: towards community cloud computing. In *Digital Ecosystems and Technologies, 2009. DEST'09. 3rd IEEE International Conference on*, pages 103–108. IEEE, 2009.

[15] Rajkumar Buyya, Chee Shin Yeo, and Srikumar Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on*, pages 5–13. Ieee, 2008.

[16] Alfredo Cuzzocrea, Il-Yeol Song, and Karen C Davis. Analytics over large-scale multidimensional data: the big data revolution! In *Proceedings of the ACM 14th international workshop on Data Warehousing and OLAP*, pages 101–104. ACM, 2011.

[17] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, pages 137–150. USENIX Association, 2004.

[18] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[19] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE'08*, pages 1–10. Ieee, 2008.

[20] Wikimedia Foundation. Wikipedia static html dumps.

[21] Eugene Garfield, AI Pudovkin, and VS Istomin. Algorithmic citation-linked historiographymapping the literature of science. *Proceedings of the American Society for Information Science and Technology*, 39(1):14–24, 2002.

[22] National Human Genome Research Institute. The 1000 genomes project more than doubles catalog of human genetic variation, October 2012.

[23] Platform as a service (paas) drives cloud demand. Technical report, Intel, August 2013.

[24] L. Kleinrock. UCLA to be first station in nationwide computer network. UCLA Press Release, July 1969.

[25] Raffi Krikorian. New tweets per second record, and how!, August 2013.

[26] Peter Mell and Timothy Grance. The nist definition of cloud computing. 2011.

[27] Microsoft. Pricing calculator — windows azure, 2014.

[28] Microsoft. Windows azure, 2014.

[29] NICTA. Nicta/scoobi, 2014.

[30] Brendan O'Connor, Ramnath Balasubramanyan, Bryan R Routledge, and Noah A Smith. From tweets to polls: Linking text sentiment to public opinion time series. *ICWSM*, 11:122–129, 2010.

[31] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. 1999.

[32] Xiongpai Qin, Biao Qin, Xiaoyong Du, and Shan Wang. Reflection on the popularity of mapreduce and observation of its position in a unified big data platform. In *Web-Age Information Management*, pages 339–347. Springer, 2013.

[33] salesforce. Salesforce crm, 2014.

[34] Y. Shafranovich. Common format and mime type for comma-separated values (csv) files, October 2005.

[35] Chris Snijders, Uwe Matzat, and Ulf-Dietrich Reips. "big data": Big gaps of knowledge in the field of internet science. *International Journal of Internet Science*, 7(1), 2012.

[36] Ronald C Taylor. An overview of the hadoop/mapreduce/hbase framework and its current applications in bioinformatics. *BMC bioinformatics*, 11(Suppl 12):S1, 2010.

[37] Twitter. twitter/scalding, 2014.

[38] Luis M Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, 2008.

[39] Daniel Warneke and Odej Kao. Nephele: efficient parallel data processing in the cloud. In *Proceedings of the 2nd workshop on many-task computing on grids and supercomputers*, page 8. ACM, 2009.

[40] Daniel Warneke and Odej Kao. Exploiting dynamic resource allocation for efficient parallel data processing in the cloud. *Parallel and Distributed Systems, IEEE Transactions on*, 22(6):985–997, 2011.

[41] Tom White. *Hadoop: The Definitive Guide: The Definitive Guide*. O'Reilly Media, 2009.

[42] Tom White. The small files problem, Febuary 2009.

[43] WLCG. Welcome to the worldwide lhc computing grid — wlcg.

[44] Paul Yang. Moving an elephant: Large scale hadoop data migration at facebook, 2011.

[45] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 10–10, 2010.

[46] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1):7–18, 2010.

[47] Weizhong Zhao, Huifang Ma, and Qing He. Parallel k-means clustering based on mapreduce. In *Cloud Computing*, pages 674–679. Springer, 2009.

Appendix A

Personal Reflection

At the beginning of my degree I found it difficult to imagine dedicating such a large amount of time to one topic.

I feel one of the key factors in finishing the project successfully was following an Agile approach. Reviewing the work completed on a weekly basis helped make more realistic estimates for future work, and let to more realistic scheduling. In order to facilitate this, I'd recommend other students to provide a relatively loose schedule in the early stages of the project. It is likely that the (feasible) requirements for the project will become more clear as the project progresses, and having a flexible schedule allows for changes as things become clearer. Having a loose schedule also allows for adjustments when things go wrong. Inevitably tasks will take longer than expected and work will appear which was not anticipated.

The Final Year Project provides very strict milestones in terms of the deadlines imposed on students. My experiment design was modular in nature, so certain experiments could be cut or simplified when there was not enough time to implement the original plan. The ability to cut scope whilst still meeting the minimum requirements is essential in a project of this nature, with very strict, immovable deadlines.

Maintaining this methodology relies on the student having discipline to set clear goals each 'sprint', and to review the progress towards those goals at the end in order to accurately ascertain what amount of work is feasible for each time period. For the Final Year Project I found that a week was naturally suited to being the length of a sprint, as supervisor meetings provided an opportunity to reflect on progress and set goals. It is much easier to set realistic expectations in the short term, rather than over the length of the whole project.

Taking the time to set out a strong set of Aims & Objectives at the beginning of the project was something which served me well, as it provided a clear end goal for the project. While the method of getting there changed, having a core direction to aim toward was helpful.

From a technical perspective I would encourage students undertaking Big Data projects to be wary of the length of time it takes to correctly configure Hadoop and similar tools. They are far more complex than they initially appear; this was the cause of the largest deviation from my project schedule. Similarly, I found several problems with libraries and frameworks. In particular, working with a beta version of Stratosphere caused significant problems at points. I would encourage future students to work on stable branches of software whenever possible, and to strongly consider whether the move to a less stable version is worth the additional features.

Using version control helped greatly throughout the project, allowing me to experiment with the experiment code without worrying about losing any prior, working code. It also made it much simpler to work on any machine, and ensured that the project was well backed up throughout.

I would strongly encourage future students to take a project in an area of interest, and to choose a topic which will challenge them. The Final Year Project provides an excellent opportunity to learn new topics and work in interesting areas in much more depth than the rest of the degree program.

Appendix B

Record of External Materials Used

- Data was acquired from the Wikimedia Foundation, who provide semi-reguar HTML dumps of various language versions of wikipedia [20], under the Creative Commons license.
- Software developed for the Hadoop platform used the Scoobi [29], a library developed by NICTA.
- The Hadoop PageRank implementation was modified from code examples provided by the Scoobi [29] project.
- The Stratosphere PageRank implementation was modified from code examples provided by the Stratosphere project.

Appendix C

Potential Ethical Issues

No ethical issues arose during my project.

Appendix D

Sample Code: Hadoop Reverse Link Graph Implementation

```
/**
 * Find Reverse Link Graph using Hadoop
 */
import com.nicta.scoobi.Scoobi._
import org.apache.hadoop.io.Text
import org.jsoup.Jsoup

object ReverseLinkGraph extends ScoobiApp {
  def run() {
    val data: DList[(Text, Text)] = fromSequenceFile[Text, Text](args(0))
    val result = data.mapFlatten(in =>
        {
                val key = in._1.toString
                val body = in._2.toString
                val addresses = findLinkDestinations(body)
                addresses.map(a => (a, key))
        })
        .groupByKey

    persist(result.toTextFile(args(1)))
  }

  /**
  * Find all link tags in a document and return the page they link to
  * @param html The HTML document to find links in
  * @return A list of strings representing link destinations
  */
  def findLinkDestinations(html: String): Iterable[String] = {
    val links = Jsoup.parse(html).select("a[href]").iterator().toList
    links.map(l => l.attr("href").split("/").last)
  }
}
```

Appendix E

Sample Code: Stratosphere Reverse Link Graph Implementation

```
import java.util.Iterator;
import java.util.StringTokenizer;

import eu.stratosphere.api.common.JobExecutionResult;
import eu.stratosphere.api.common.Plan;
import eu.stratosphere.api.common.Program;
import eu.stratosphere.api.common.ProgramDescription;
import eu.stratosphere.api.common.operators.FileDataSink;
import eu.stratosphere.api.common.operators.FileDataSource;
import eu.stratosphere.api.java.record.functions.FunctionAnnotation.ConstantFie
import eu.stratosphere.api.java.record.functions.MapFunction;
import eu.stratosphere.api.java.record.functions.ReduceFunction;
import eu.stratosphere.api.java.record.io.CsvOutputFormat;
import eu.stratosphere.api.java.record.io.TextInputFormat;
import eu.stratosphere.api.java.record.operators.MapOperator;
import eu.stratosphere.api.java.record.operators.ReduceOperator;
import eu.stratosphere.api.java.record.operators.ReduceOperator.Combinable;
import eu.stratosphere.client.LocalExecutor;
import eu.stratosphere.types.IntValue;
import eu.stratosphere.types.Record;
import eu.stratosphere.types.StringValue;
import eu.stratosphere.util.Collector;

import scala.Tuple2;

/**
 * Take a group of web pages, and compute the Reverse Link Graph
 * (find a list of pages which link to any other given page)
 */
public class ReverseLinkGraph implements Program, ProgramDescription {

        private static final long serialVersionUID = 1L;


        /**
         * Takes a record representing a web page, and pushes a list of link p
         * collector. Link pairs are (dest, src) where the source is the curre
         * and the destination is the place the links are going to.
         */
        public static class GetLinks extends MapFunction {
                private static final long serialVersionUID = 1L;

                @Override
                public void map(Record record, Collector<Record> collector) {
                        // get the first field (as type StringValue) from the 
                        String line = record.getField(0, StringValue.class).get

                        // parse the CSV and remove quote escaping for HTML pa
                        String[] parts = line.split(",");
                        String key = parts[0];
                        String value = "";

                        for(int i = 1; i < parts.length; i++) {
                            value += parts[i] += ",";
                        }

                        // perform actual maps
                        scala.collection.Iterable<Tuple2<String, String>> scala
                scalaPairs = AshleyIngram.FYP.Core.ReverseLinkGraph.map

                // Convert to Java collection to iteration
                Tuple2<String, String>[] pairs = new Tuple2[scalaPairs.
                scalaPairs.copyToArray(pairs);

                // output from map
                for(Tuple2<String, String> pair : pairs) {
                        collector.collect(new Record(new StringValue(pa
                }
        }
}

/**
 * Groups links by key (the destination), merging the source links into
 * ((dest, src1), (dest, src2)) becomes (dest, "src1, src2")
 */
@Combinable
@ConstantFields(0)
public static class GroupByKey extends ReduceFunction {

        private static final long serialVersionUID = 1L;

        @Override
        public void reduce(Iterator<Record> records, Collector<Record>
                Record element = null;
                String result = "";

                while (records.hasNext()) {
                        element = records.next();
                        // Add the link source and a comma (source1, so
                        result += element.getField(1, StringValue.class
                        result += ",␣";
                }

                result += "\n";

                // Change the second value in the record to be the com
```

```
                        element.setField(1, new StringValue(result));

                        // output to the world!
                        out.collect(element);
                }

                @Override
                public void combine(Iterator<Record> records, Collector<Record>
                        // the logic is the same as in the reduce function, so
                        reduce(records, out);
                }
        }


        @Override
        public Plan getPlan(String... args) {
                // parse job parameters
                int numSubTasks   = (args.length > 0 ? Integer.parseInt(args[0]
                String dataInput = (args.length > 1 ? args[1] : "");
                String output    = (args.length > 2 ? args[2] : "");

                FileDataSource source = new FileDataSource(new TextInputFormat(
                MapOperator mapper = MapOperator.builder(new GetLinks())
                        .input(source)
                        .name("Get␣Links")
                        .build();
                ReduceOperator reducer = ReduceOperator.builder(GroupByKey.clas
                        .input(mapper)
                        .name("Group")
                        .build();
                @SuppressWarnings("unchecked")
                FileDataSink out = new FileDataSink(new CsvOutputFormat("\n", "

                Plan plan = new Plan(out, "Reverse␣Link␣Graph");
                plan.setDefaultParallelism(numSubTasks);
                return plan;
        }
        @Override
        public String getDescription() {
                return "Parameters: <numSubStasks> <input> <output>";
        }

        public static void main(String[] args) throws Exception {
                ReverseLinkGraph rlg = new ReverseLinkGraph();

                if (args.length < 3) {
                        System.err.println(rlg.getDescription());
                        System.exit(1);
                }

                Plan plan = rlg.getPlan(args);
                JobExecutionResult result = LocalExecutor.execute(plan);
                System.err.println("Total runtime: " + result.getNetRuntime());
        }
}
```