MSc Dissertation

A Map/Reduce Framework for Evaluation in Machine Learning

Karl OBrien

A thesis submitted in part fulfilment of the degree of

MSc in Advanced Software Engineering

Supervisor: Padraig Cunningham

UCD School of Computer Science and Informatics
College of Engineering Mathematical and Physical Sciences
University College Dublin

December 13, 2012

Table of Contents

Abstract

Machine Learning is emerging as a huge growth area in the expanding world of understanding patterns in large amounts of data. A typical limiting factor in machine learning is the validation process where the generated results are examined for errors and accuracy. One such example is Model/ Parameter Selection as choosing the optimal parameters to apply against a dataset can be painstaking process involving testing the same dataset multiple times to determine the best value. The MapReduce model provides a framework to allow the evaluation process to be carried out in parallel - testing out multiple values at the same time.

In this report, a parallel framework focusing on the testing phase of machine learning algorithms is presented. Intensive tests were run against (Naive Bayes and J48 algorithms) to determine the speed-up provided by the framework.

The report demonstrates that the system scales well by implementing the Cross Validation process in parallel, achieves significant speedup over the traditional sequential cross validation process. It is also a flexible platform which can be modified to search for optimal K values in nearest neighbour algorithms as outlined in the report.

Acknowledgements

I would like to thank Professor Padriag Cunningham for providing the idea behind this thesis and for his advice and encouragement throughout the project. I would also like to thank Annmarie for all her support and my parents for always backing me in everything I do.

Chapter 1: **Introduction**

1.1 Background to Machine Learning

With the explosion of online services, cheap disks and always available online storage coupled with internet access that is getting faster all the time, it has become very easy to simply save items which would have previously been thrown away. Items are being recorded about customer movements, loyalty cards for shopping in supermarkets, recommender systems for online shopping and internet search history. This area of research is typically referred to as Data Mining. This is defined as the process of discovering patterns in data [1]. In most cases when looking at a large volume of raw data, it is very difficult to find meaning in the data but if the data was broken down into categories or groups that do have meaning then a clear picture starts to emerge from the data. Examples include looking for certain words or terms from spam filters or certain patterns for handwriting recognition. Machine Learning is a key process in turning mass amounts of data into useful information. It sits at the intersection of multiple disciplines (Engineering, Computer Science and Statistics).

More and more professions are turning towards this subject to try and interpret large amounts of data in a quicker way.[2] A popular example of a machine learning algorithm that has made its way into the mainstream is Google's page-rank algorithm. [3] This has changed the way data is processed over the internet. This algorithm has been listed as one of the top 10 most influential algorithms. [4]

While the algorithms for finding patterns and meaning in this data are being constantly researched and implemented in more efficient ways, one item to remember is that in most cases the most simple classification rules perform the best on the most common datasets. [5] This report is concerned with looking at the process of evaluating the data that is already gathered and investigating a more scalable solution to the calculation of errors in the evaluation process. If there are multiple algorithms used against a particular data-set; which algorithm is the best fit against the data? In other words, how is the accuracy of the data evaluated? It is one thing to take meaning from a set of data but the process of validating that the inferred data is correct is a challenging process.

The examples provided above prove that there is an ongoing demand to extract meaning from data and this is turn brings up the issue of scalability. In the past crunching large amounts of data was the domain of high performance computing. These systems were typically built for purpose tools that were becoming harder and harder to manage. Indeed [6] outlines in his report how a lot of the HPC architectures were not suited to solving a lot of scientific problems.

1.2 Issues

There are a couple of different scenarios when trying to evaluate data. In the first case, there is an abundance of quality data. Training can be carried out on one section of the data and testing can occur on another section of data. This is an ideal situation as there is enough data for separate training and testing. The second case would be where there is a lack of quality data. This means that there is a lack of data for both training and testing independently. An important fact is that error rates on training data is not likely to be a good indicator of error rate on new data [1].

1.2.1 Cross Validation

Cross Validation is a very popular framework for estimating errors on limited datasets. For K = 10, this involves randomly partitioning the training set into training (90% of original training set) and validation (10% of original training set). In this way each partition gets used for testing while the remainder is used for training. At the end of the process the error rate from each validation set is averaged to provide the overall estimate. Schaffer [7] outlines how cross validation can work in certain cases to choose a classification algorithm. When choosing CV, Kohavi [8] outlines how 10 fold cross validation in most cases is the most efficient method for cross validation.

1.2.2 Parameter Selection

Parameter selection on the nearest neighbour classification algorithm. Choosing the value of the K parameter can be a time consuming process and it is another element that can be carried out in parallel. Again Kohavi [9] outlines an algorithm for enhancing error estimation. The subject of finding the best parameter to fit a dataset is an interesting problem and a method of automating this process is discussed here. [10]

1.2.3 Scalability

The overhead of dealing with large amounts of data combined with data intensive processes has led to a need to look at scalable systems, that is architectures that can handle increased workload with reasonable performance [11]. When discussing scalability, one platform that lends itself to this topic is that of MapReduce [12]. The performance of this programming paradigm and its benefits when applied to data intensive scientific analysis have been the subject of recent investigation [13] and would appear to be well suited to the overheads in choosing values for cross validation and parameter selection.

Looking at the definitions of CV and Parameter selection, both methods involve carrying out repeated tests across the dataset. In the case of CV, there is a need to divide up the data multiple times, then evaluate and test the results. This can be a very time consuming operation. For example, running a J4.8 decision tree learner on a dataset [1] took over two hours to build its output. If this was applied to 10 fold cross validation, this process will need to be carried out 10 times and then the error rates averaged together. Often in order to find the best value in a nearest neighbour algorithm, different parameters of K are tested to see which value performs best. Again this is a sequential operation. For example, K Nearest Neighbour can be a another time consuming process [2]. Both methods described above are open to a parallel architecture. This report will look closely at how running these methods will scale from a hardware and software point of view.

[1] KDD cup data 1999 - full set - 710MB- running on a Dell R610, 32GB Ram

[2] KDD Cup 1999 10% - 6 hours

1.3 Aims of the project

1. Build a parallel architecture for CV and Parameter Selection. This project will outline an architecture using Weka [1] (an open source machine learning tool) and Hadoop [14] (map-reduce framework). The current Weka implementation can be run on the command line or through a graphical user interface. One of the main issues with Weka is the fact that is carries out its validation checks in a sequential manner, this adds to the run time of the platform as it needs to run several times to evaluate the data.
2. To document and explore the the speed-ups provided by the architecture. This will take into account the limitations of running the algorithm in parallel, tuning the platform to meet the requirements of running in parallel. Finally there will be a statistical analysis of the results of running the system using the standalone Weka suite as the benchmark.

Please note that while the experiments involve passing a body of data through machine learning algorithms, the accuracy of the algorithms is not the main goal but rather a discussion on how the system scales to meet the demands of passing this data through in a parallel format.

1.4 Report Structure

The document is broken down into the following sections:

- Chapter 2: Problem Domain

  There will an introduction to machine learning and evaluation techniques. The difficulty in implementing a one size fits all solutions to each dataset. There will be a particular focus on historical methods in trying to implement these techniques, ongoing efforts to improve the evaluation process. There will also be a section on scalable software and how hardware also has a large part to play in any possible solutions. Tuning the environment is also a very important process and there will be a discussion on various techniques of building a system that can perform to a known capacity and workload There will be a brief look at large datasets and the scalable architectures that are being built to deal with the different requirements of always online data.
- Chapter 3: Weka and Hadoop Platforms:

  There will be a discussion on the features of both the Weka and Hadoop platforms, the strengths and weaknesses associated with each.
- Chapter 4: Platform Architecture:

  A full description of the platform built to demonstrate the scalability of machine learning error estimation techniques using different input values.
- Chapter 5: Results/ Experiments:

  Performance and statistical analysis of the results of running a number of different scenarios across a specially built lab environment for this system.
- Chapter 6: Conclusions:

  A discussion based on the experiments and opportunities for future work in this environment.

Chapter 2: **The Problem Domain**

In this section, a background is provided on typical data intensive applications and scalability issues. MapReduce is introduced as a possible solution to reducing load across systems. Machine Learning is also discussed - this topic provides lots of opportunity for testing how systems scale mainly because of the volume associated with such platforms. Another reason to look at this area is the computational complexity associated with running the algorithms. This provides an excellent opportunity to benchmark common datasets against each architecture. There is also a section on scalable computing with emphasis on options for tuning both hardware and software to handle data intensive operations.

2.1 Data Intensive applications

The definition of data intensive applications provided by [15] [1].

Applications that explore, query, analyse, visualise, and, in general, process very large scale data sets are known as Data Intensive Applications

The growth of data in the digital age has been astonishing. The amount of data that is generated each day is growing at a massive scale. An entry in the Science Journal calculated that the amount of data stored in the world as 295 Exabytes. [2] This number is only going to increase in the future. A good example of how much data usage is increasing is looking at world wide electricity usage in data centers. According to [16], direct electricity used by information technology equipment in data centers represented about 0.5% of total world electricity consumption in 2005. Organisations that were not typically known for dealing with large amounts of data now find themselves in a situation where in the first case they need to save a lot of their transactions online for legal requirements and in the second case there is a lot of useful data contained in their online data. These same companies are starting to ask for the first time if there is anything in their data that will tell them more about their customers and of course they want this data in the quickest most efficient manner. [17] describes the problems that face engineers building data intensive systems today. Issues such as massive datasets, the need for flexible, always on systems that provide high performance and reliability for data. There is a limit to the amount of hardware that can be used and so the focus turns to using the environment that is already in place in a better way. Given that all transactions are moving to an online environment, the need to build more efficient applications, scale applications in the correct environment is becoming more and more important. A key ingredient to building massively scalable and data intensive applications is merging hardware and software together effectively.

[1] http://www.inf.ed.ac.uk/publications/thesis/online/IM100765.pdf, definition also contained in this thesis
[2] http://www.bbc.co.uk/news/technology-12419672

2.2 Machine Learning

As mentioned in Chapter 1, machine learning is an automated method of pulling patterns or meaning from data. The field seems to be a perfect fit when discussing data intensive applications and multi-core systems. In this field there are a number of well known algorithms for gaining an insight from volumes of data. Combine this with improved raw computing power and there are infinite possibilities for what a research body or company might be able to achieve. An indication of how important this field will become was outlined by Hal Varian, chief economist at Google.

" You also want to be able to visualize the data, communicate the data, and utilize it effectively, being able to access, understand, and communicate the insights you get from data analysis are going to be extremely important." [3]

2.2.1 Key Steps in Machine Learning

In order to successfully gain insight from a large amount of data, it is not simply a case of throwing large amounts of data at random algorithms and waiting to get the results back from the system. There are a number of steps to follow in order to make sure that the feedback from the machine learning process is correct. The best practises in Machine Learning are outlined in [1] and [18],

1. Collect Data

   This might be publicly available data [4]. It could be data that a company or institute gather themselves.

2. Prepare the Input Data

   For the purposes of this report the data format that will be used is the standard Weka version, ARFF [5]

3. Analyse the Input Data

4. Train the Algorithm

5. Evaluate the Algorithm

   The evaluation section which is part of the testing section will become the focus of the report in later chapters. Since this report will concentrate on classification algorithms, the best method to measure the performance of a classifier is to measure the error rate.

2.2.2 Key Evaluation Methods

Cross Validation

Figure 2.1 outlines the Cross Validation process. Train all the folds but the Kth and test on the Kth to estimate error.

$\sum_{K=1}^{5} ERR_k$

[3]McKinsey Quarterly, January 2009

[4]UC Irvine maintain a repository for machine learning @ http://archive.ics.uci.edu/ml/index.html

[5]http://www.cs.waikato.ac.nz/ml/weka/arff.html

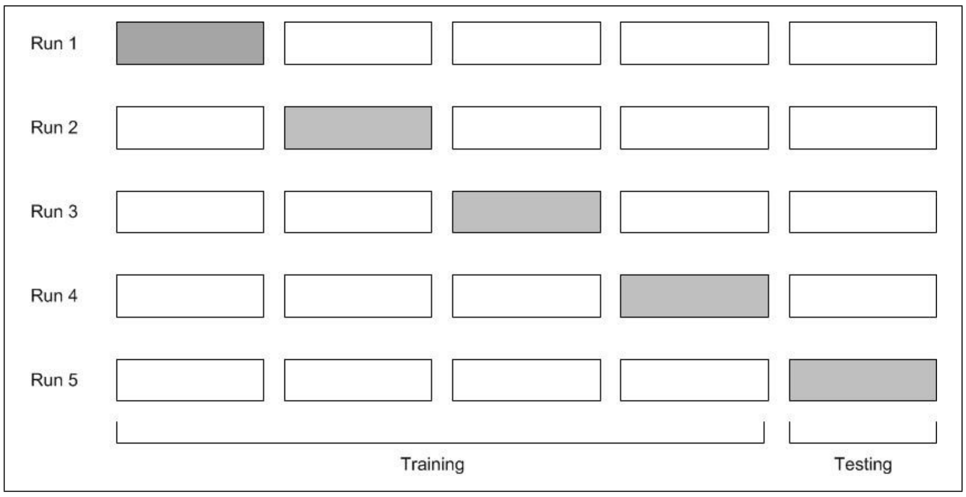

Figure 2.1: Cross Validation Process

Parameter Selection

The process of parameter selection is not an formulaic process like cross validation but dictated by prior history and experience in the typical values that work well against particular sets of data. In a lot of cases, the process will be formulated by certain rules that are suggested by the author of the machine learning algorithm. Kohavi [10] proposes a wrapper method around the training algorithm that will automate the process of choosing the best values for the algorithm. This is an interesting proposition in that if it is possible to place the algorithm into a parallel architecture, then it would be possible to simultaneously calculate different parameters and compare the results to find the minimal error.

2.3 Scalable Computing

"Scalable Computing is defined as the ability of a system to continue to meet its response time or throughput objectives as the demand for the software functions increases." [19]

An example of this topic is outlined in the sample graph

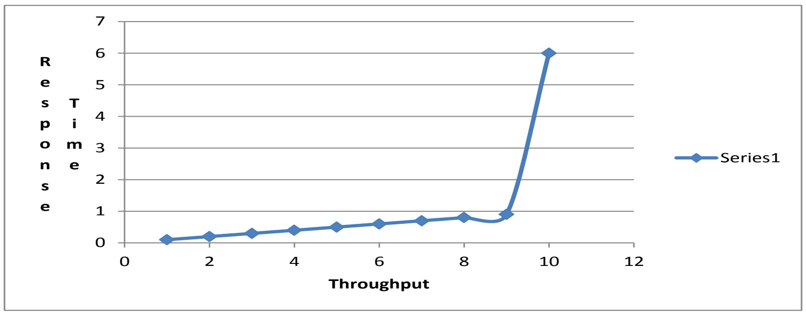

Figure 2.2: Throughput example

In Figure 2.2, when plotting throughput against response time, increases in throughput do not have an effect on response time until the system hits a critical point where the response time curves upward. This is known as a system bottleneck. Scalability is concerned with knowing when a system will hit its bottleneck and under what conditions. The earlier topics in the chapter explain where the need for large scale systems that can process and apply machine learning using machine learning algorithms is coming from. The following section lists some of the systems and techniques that have come to the fore in order to process this data in the most efficient way. Often these methods cleverly break data down into more manageable parts so the volume is easier to deal with. Bryant [6] outlines the four key principles that data intensive, scalable computing systems will need to exhibit.

1. Intrinsic rather than extrinsic data

   It should be the systems job to collect and maintain the data. It is not the job of the individual users. This means that the system can build in controls such as availability and error correction to manage the data. From this users will be able to query the data.

2. High Level programming model for expressing computations over data

   It is not the job of the programmer or user to decide how many jobs or nodes to use to process the data. This should be left to the system to decide.

3. Interactive access

   The users should be able to interact with the underlying system. I.e. they should be free to run their own jobs on the system without needing to line up resources. This is where it is crucial that the system is able to scale to meet the demands of all users.

4. Scalable Mechanisms to ensure high reliability and availability

   The system should be available 24 x 7 and should deal with items like disk failures without downtime. It should also be able to recompute jobs again in the event of failures.

2.3.1 Multi-core Systems & MapReduce

Throughout the 70's, 80's and 90's, there was a race towards fast clock speed on micro-processors in line with Moore's Law [20]. Since the turn of the century due to heat and power issues, there has been a marked move towards multi-core systems in mainstream computing. Gorder [21] outlines how multi-core processors have now gone mainstream but the software side will now need to catch up on the hardware. To make the most of the new hardware, there is a need for new tools, new algorithms and a new way of looking at programming.

Sutter and Larus[22] point out that multi-core mostly benefits concurrent applications, meaning one where there is little communication between cores. There are huge benefits to this hardware architecture if the data is divided and stays local at the cores.

One technology that has come to the fore in the past ten years is a method that was inspired by a Google paper [23] outlining how Google took something like MapReduce which has been around for years and made it into a computing platform capable of handling very intensive operations. It origins can be traced back functionality present in Lisp and other functional programming languages [23]. A key ingredient of MapReduce is that it allows a programmer to break the problem down into smaller parts. According to [24], it is possible for a future programmer to speed up machine learning algorithms by adding more cores to the problem rather than needing specialised software/ hardware. This is a very important contribution to the notion of scalable computing. This technology is focused on commodity hardware that runs multiple cores - this fact makes it possible to achieve linear speed-up on non-specialised hardware.

The combination of MapReduce on multicore systems has been recognised and there are a number of interesting uses of the platform. [24] is outlining how machine learning algorithms can be converted to execute entirely in a MapReduce framework. [25] describes how it is reaching out to a wider variety of uses with references to earthquake simulation and image processing tasks. Clearly it is suited to solving a diverse array of problems all with data intensive processing as the common factor.

Listing 2.1: MapReduce Diagram.

```
map(input_record) {
    ...
    emit(k1, v1)
    ...
    emit(k2, v2)
    ...
}

reduce (key, values) {
    aggregate = initialize()
    while (values.has_next) {
        aggregate = merge(values.next)
    }
    collect(key, aggregate)
}
```

Listing 2.1 [6] displays a typical MapReduce set-up where the map is separating the data into keys and values which are then grouped together by the reduce process.

Notable systems for MapReduce include Hadoop [7] and Disco [8]. It is interesting to note that both system conform to the conditions laid out by Byrant [6]. Mahout [9] is another interesting platform in that its goal is to convert machine learning algorithms to use the MapReduce model. Its reach has even extended into the database world with systems such as mongodb [10] and couchdb [11]. All the above systems are operating in large scale operations in production settings working with large amounts of data.

2.3.2 Parallel Systems Performance Techniques

While the technology behind the latest micro-processors is constantly changing with multi-core architectures joining the mainstream and highly efficient algorithms such as map reduce being applied to huge datasets, there is a need to evaluate the impact or performance improvement that these elements bring to a system. New cloud computing environments now mean that being able to estimate the load and scale of future data also means being able

[6]http://architects.dzone.com/articles/how-hadoop-mapreduce-works
[7]http://hadoop.apache.org
[8]http://discoproject.org
[9]http://mahout.apache.org
[10]http://www.mongodb.org
[11]http://www.couchdb.org

to estimate costs associated with this data [26]. Some common techniques [27] are outlined below that help to judge how well a system will respond to more resources.

- Parallel Speed-up

  Speedup is commonly defined as the ratio between the total elapsed time on one processor divided by the total elapsed parallel execution time on all processors.

$$Speedup(m) = \frac{T(1)}{(T(m)}$$

- Scale Up

  Scale-up is defined as measuring how well the parallel system reacts to the growth of both the system and the data size.

$$Scaleup(m) = \frac{T(1, D)}{(T(m, mD)}$$

- Size Up

  Size-up is defined as the ability of the system to handle data growth.

$$Sizeup(m, n) = \frac{T(m, nD)}{(T(m, D)}$$

Performance Tuning

An important element of systems that are able to handle a growing amount of data in a capable manner is the set-up of the runtime of the environment. This can refer to both the set-up of the hardware and the software of the system. The environment can scale vertically (this involves adding more CPU power or memory to a single node) or scale horizontally (simply adding more nodes to the system).

Hardware Tuning

In order to prepare a computing system for handling data intensive loads, there are a couple of areas that can be looked at. In terms of this report the most important points are listed below but a more complete guide is available here. [12]

1. Interrupt and Process Binding

   On multicore systems, it is possible to bind the application threads to specific CPU cores. This can eliminate time spent migrating tasks between the processors. It also ensures that the application data remains in the CPU caches. Common approaches to this problem involve using command such as "taskset" and "cpuset". If there is a certain task that is bound to a cpu, then it is also possible to prevent certain interrupts [13] from being serviced on that particular processor. In effect the cpu is being isolated from the rest of the system.

2. Tuning Process Priority

   It is possible to assign more CPU to a process by lowering its "nice" level. Setting the level to -20 will ensure that it will be running at the highest priority

[12]http://www.redbooks.ibm.com/redpapers/pdfs/redp4285.pdf

[13]http://www.6test.edu.cn/ lujx/linux_networking/0131777203_ch02lev1sec2.html

3. Processor

   Many modern processors include a turbo mode which can be enabled to allow overclocking to activate in the event of power or thermal events in the processor. In the case of data intensive applications this can be a useful feature.

Software Tuning

Since this report will focus on open-source frameworks that were built using the Java programming language, it is necessary to mention some of the background in tuning Java systems. As in the hardware tuning section, there are a couple of key facts to remember when scaling java systems.

1. Heap Space

   Each time a JVM is started it requests and gets some memory from the operating system, this is called the heap space. Each time an object is created, it is allocated from the heap space or when the object dies, the memory returns to the heap space. Most applications can use the default values that come with each JVM, but in the case of data intensive applications and especially machine learning algorithms, allocating extra memory towards the heap is a must.

2. Garbage Collection

   Garbage Collection is the process of reclaiming heap space from objects that are available for collection. In terms of a scalable system, it is important to understand how the system will perform its garbage collection. It is possible for this to become a bottleneck of the system. If there is a memory leak, the garbage collection thread will need to run all the time which can lead to an unresponsive system.

3. Stack Size

   Each thread in the JVM will get a stack. Again it is possible to run out of memory if the number of threads allocated exceeds the size of the stack.

Each of the topics in both the hardware and software section are worth keeping in mind when looking to build or modify a system that will process data intensive tasks. Given that that this report will examine a Java framework for MapReduce and execute machine learning tasks on this framework through a Java API, the points above will become very relevant.

Chapter 3: **Machine Learning and MapReduce platforms**

This section will introduce the Weka and Hadoop platforms that were used to built the system to evaluate machine learning in parallel. It will look at the building blocks for each platform, discussing the various parts that enable the system to function.

3.1 Weka as a Platform

The Weka system provides an out of the box solution for applying machine learning algorithms to datasets. It is developed and maintained by University of Waikato, New Zealand. Its goal is to take a set of data and provide its final representation as a set of rules or in a decision tree format. It will allow a user to carry out the steps provided in section 2.2.1 without needing to write any code. It is especially useful for tweaking parameters to determine which conditions suit the applied dataset the best. While this approach works on smaller datasets, changing parameters such as the cross validation fold (in any algorithm) or the K value (nearest neighbour) can take hours to test. Weka can run in a number of different modes: GUI Explorer, command line and directly through Java code. For the purposes of integrating the system into a parallel architecture, this project will utilise Weka directly through its Java API. For the bench marking section in chapter 5, the command line instance of weka will be used as it is more suited to testing larger datasets due to memory issues with the GUI version.

3.2 Approach to machine learning

Weka follows the key steps in machine learning. In order to use the system there must first be a dataset. The data can be inserted into Weka in a number of ways: csv or Attribute-Relation File Format. [1]. Fig 3.1 shows the flow of the Weka software as it moves through the stages of processing the data.

3.2.1 Command Line

In order to use weka from the command line the following outlines the ease of running the tool

```
java -cp weka.jar -Xmx4096m weka.classifier.trees.J48 -t iris.arff -x 10
```

The value of x is very important in the example as this specifies the number of cross validation

[1] http://weka.wikispaces.com/ARFF

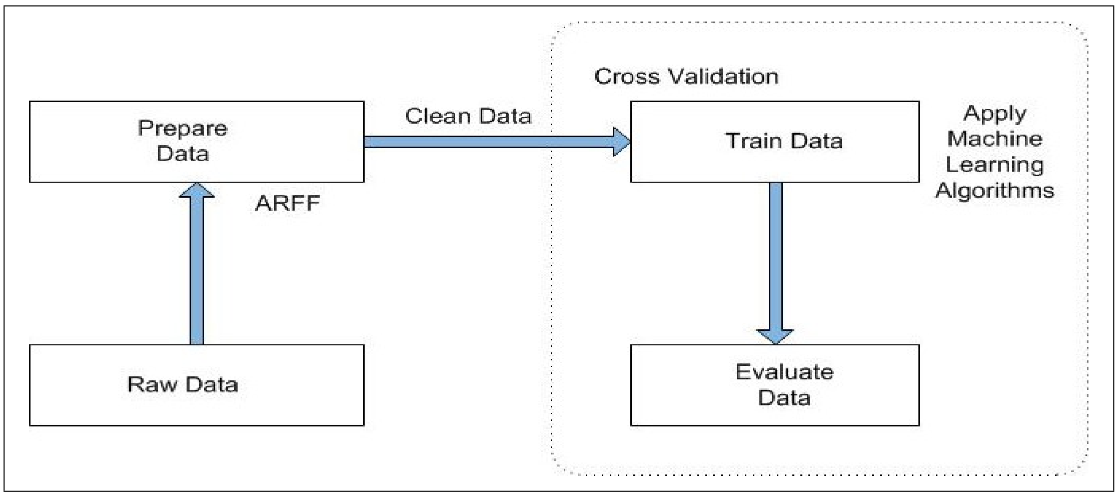

Figure 3.1: Outline of the Weka System

folds to carry out. In this case the tool will effectively loop through the data 10 times applying the CV algorithm each time. It will log the output of each iteration and its final output will be the average error rate.

3.2.2 Cross Validation

The internal process that Weka is using outlined through the code extract below. An important point to note from the extract below is the fact that weka needs to load the entire dataset into memory in order to train and evaluate the dataset. 10 folds means 9/10's of the data is trained and the remaining 1/10 is held out for testing (evaluating) the data.

Listing 3.1: Weka Source Code for Cross Validation Process.

```
int folds = 10;
   Evaluation eval = new Evaluation(randData);
   for (int n = 0; n < folds; n++) {
     Instances train = randData.trainCV(folds, n);
     Instances test = randData.testCV(folds, n);

     // build and evaluate classifier
     Classifier clsCopy = Classifier.makeCopy(cls);
     clsCopy.buildClassifier(train);
     eval.evaluateModel(clsCopy, test);
   }
```

This method will become an important part of how the parallel architecture will split the cross validation process. The fact that weka provides the functionality to perform the CV steps means that it is possible to use Weka to process the data in a parallel manner

3.2.3 Parameter Selection

Another aspect of machine learning is the ability to choose which parameters work best with a dataset. Again weka can provide the functionality to train and test the data but in a lot of cases, it can be trial and error to find out what are the best parameters that suit a dataset. Take the K-Nearest Neighbour (KNN) algorithm for example, there is no optimal choice for the value of K. Weka provides the functionality to use the KNN algorithm against a dataset.

```
java -cp weka.jar -Xmx4096m weka.classifier.lazy.IBk -t iris.arff -x 10 -K 1
```

The example will run the KNN algorithm and an object will simply be assigned to the class of the closest neighbour (K =1). A common method to find the optimal value of k would be would be run the above test multiple times, recording the error value and then using the best value. This type of method is open to a parallel architecture.

3.2.4 Scaling Weka

Although Weka is a very useful tool for applying machine learning algorithms, it is typically used on smaller datasets. The fact that the system is carrying out its CV in sequence can mean very long run times. It also loads the entire dataset when it is training a set of data, this means that certain settings needs to be enabled before using the system. One item to ensure the system can handle the data is to set the Java heap size on start-up (-Xmx), this ensures that the library can call on enough memory to build the dataset.

3.3 Hadoop as a Platform

The focus of this report is to build a scalable architecture for evaluation of machine learning algorithms. While the Weka software can take care of the machine learning element, a framework is needed to allow the system to meet the demands that will be placed on it. An ideal candidate is the open-source library - Hadoop.[2] The purpose of this software is for writing and running distributed applications that process a large amount of data. In Chapter 2, there was an outline provided by Byrant [6] on the attributes that are needed for a scalable system. Hadoop is built with these theories in mind. The system is accessible, robust, scalable and very simple to program against. Hadoop came about as a platform as a result of papers published by Google [28][12] outlining how they were scaling their search system to meet the increasing demands of the web. One element of the google papers was that they were using commodity based systems to solve their large scale problems. Their model was to use many low powered systems to act as a single distributed system. A new thought process was becoming very important in scalable systems. Instead of trying to move large amounts of data around a network to feed into high powered clusters, it was possible split the data into smaller blocks that the low powered machines could access. Thus the notion of "move the code to the data" logic was born.

[2]http://hadoop.apache.org

3.3.1 HDFS

One of the key components of Hadoop is its distributed file system - HDFS. This allows multiple Hadoop nodes to access the data in blocks and read in parallel. Its is also highly fault tolerant. Some of the feature of HDFS are outlined below: [3]

- Fault tolerance by detecting faults and applying quick, automatic recovery
- Simple and robust coherency model
- Portability across heterogeneous commodity hardware and operating systems
- Reliability by automatically maintaining multiple copies of data and automatically re-deploying processing logic in the event of failures

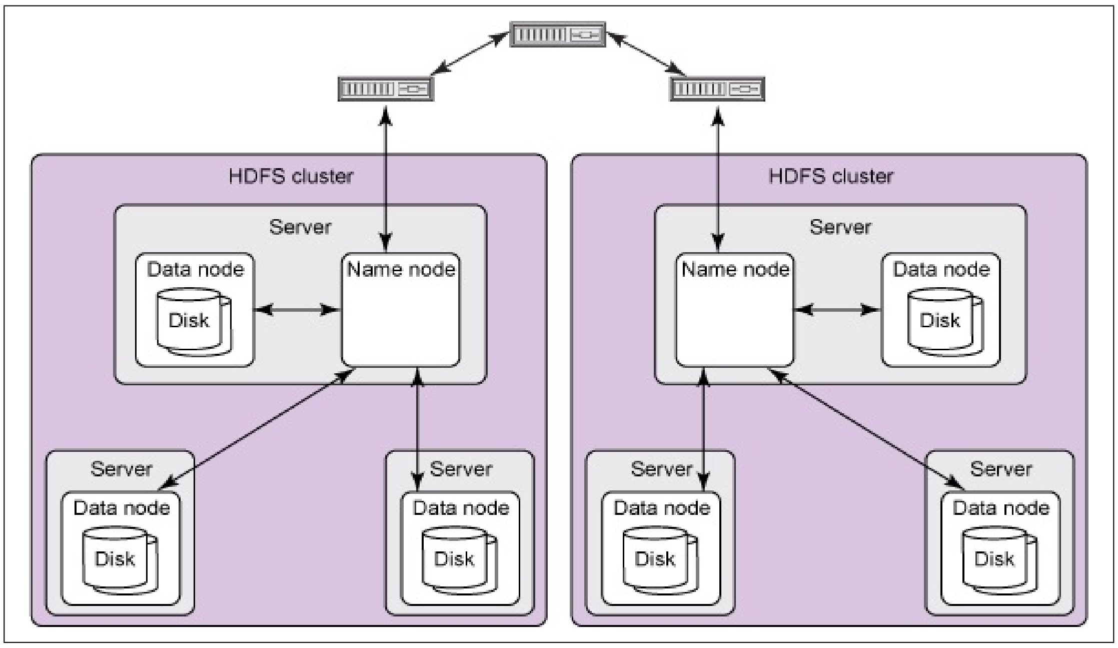

Figure 3.2: High Level Architecture of HDFS

Fig 3.2 outlines a typical HDFS container.[4] In each HDFS environment there is a component called the Name node. This will manage access to the file system and handle requests from clients. The data node will talk directly to the name node. It is in effect a Master/ Slave architecture. The name node will store all meta data for the system.

Applications that are using the HDFS are not using file systems in a traditional way. HDFS is optimised for users that made long streaming requests from the file system. It is also a block based system working on the premise that files are stored as a series of blocks. It is the job of the name node to map blocks to data nodes. The data nodes are where the data is stored. There is also a secondary name node component which in the event of failure on the main name node will provide a check point to recover the system.

[3] http://www.ibm.com/developerworks/web/library/waintrohdfs/index.html?ca=drs-

[4] http://architects.dzone.com/articles/how-hadoop-mapreduce-works

	INPUT	OUTPUT
MAP	$\langle K1, V1\rangle$	list$\langle K2, V2\rangle$
REDUCE	$\langle K2, list(V2)\rangle$	list$\langle K3, V3\rangle$

Table 3.1: Hadoop Map Reduce outline

3.3.2 MapReduce and Hadoop

With the distributed file system now in place, there is now a situation where a number of hosts have access through HDFS to a common group of files. The underlying HDFS will provide fault tolerance and ease of access to the data as well as replicating the data to where it needs to be. On top of this layer sits the data access and this is the level that programmers and developers will typically interface with in order to submit their jobs. The major advantage of this layer is that the user does not need to work with threads, locks or semaphores. In fact the user will not need to worry about shared access to data. This is where the functional model of Hadoop comes into play. Data is explicitly passed between functions which can only be changed by the function that is in focus at that particular moment.

The MapReduce methodology discussed in chapter 2 can now be applied with Hadoop. Table 3.1 outlines the various stages of mapreduce in Hadoop. The map function will transform a section of data into a number of key/ value pairs. When the map function is complete, these pairs are passed to a reduce function which sorts the data and merges the values of the same key into a single result.

Hadoop takes the theory outlined in Table 3.1 and demonstrates how MapReduce can take a large dataset that is stored in HDFS, break the data in blocks and then use a number of nodes to access different blocks of the data, run it through the map and reduce functions.

As Hadoop is written in Java, it is possible to write MapReduce functions in Java and submit them to the system. This is called the process of submitting a job. Since there is a name node and data node for managing the file system, there is a JobTracker and TaskTracker for managing the map reduce process. Perhaps the best way to imagine the system is via a diagram. [5]

Some important points to note from Fig 3.3

1. The client will submit their job to the job tracker. This will outline the functionality of the map and reduce functions. It will also specify the input and output locations
2. The job tracker will determine the number of splits that are needed.This is an important point in the development of an architecture to build weka in parallel and will be looked at in closer detail in chapter four.
3. The process will be kicked off when the job tracker contacts the task tracker which will manage the actual map process. Once finished the task tracker notifies the job tracker. (The key, value pair output is written into a memory buffer, then to file)
4. The Job Tracker then sorts the key/value pairs and notifies the necessary task tracker to invoke the reduce phase. The output is then written to file.

[5]http://architects.dzone.com/articles/how-hadoop-mapreduce-works

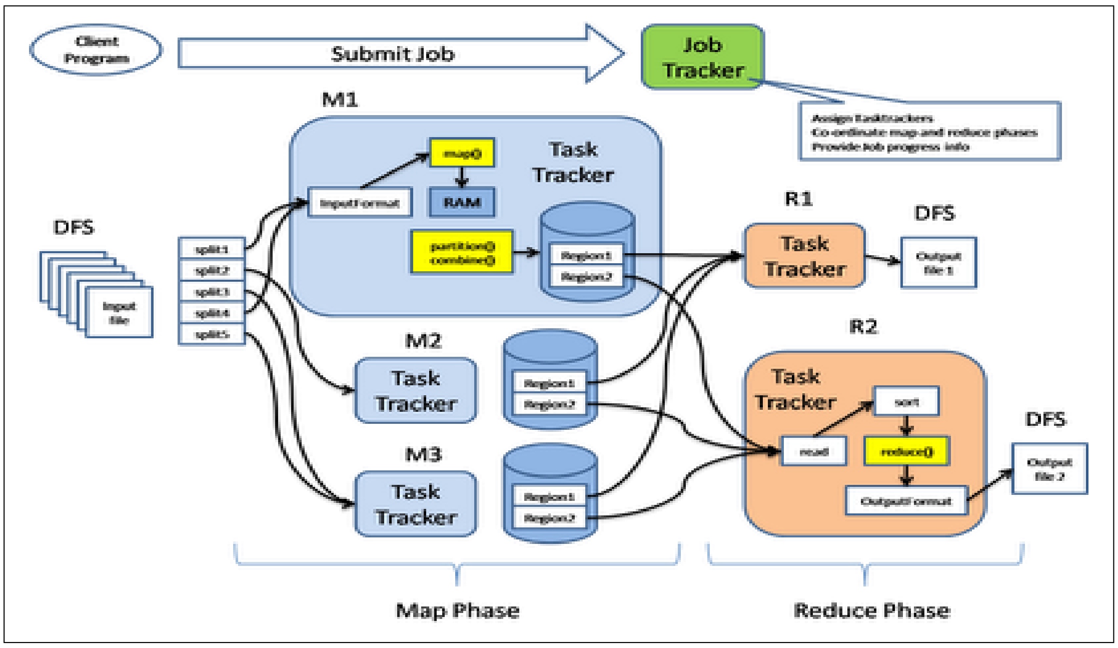

Figure 3.3: High Level Architecture of HDFS

3.4 The Life Of A MapReduce Job on Hadoop

The previous sections describe the various parts of Hadoop and it is clear that there are a lot of moving parts to consider. It would be useful to track the progress of a typical job and outline where there are performance gains to be made.

1. On Job Submission

   The framework will first look to see if any resources need to be redistributed via HDFS. For example any external files or jars that are needed by the map or reduce processes are lined up at this point. For example if a map used by external jar file for execution, this would need to be available to each node in the cluster that will be executing the task. The most important item at this stage is that the framework will need to examine the data set. This will use the Hadoop library, "InputFormat" to determine how the data will be broken down across the mapper nodes. It is at this stage that the framework will decide how many map tasks will need to be executed.

2. Map Task Submission and Execution

   At this stage the JobTracker will have a set of map execution slots available for processing (N per machine). Depending on how the data was split in the previous, each execution slot will receive a split of the input data. The TaskTracker will activate at this point, running in a child JVM to run its map task against the data. It will pass its output in key/ value form for collection by the reducer.

3. Merge-Sorting

   The JobTracker will activate the number of reduce tasks as specified by the job configuration. The keys are read from each of the mapper tasks and organised in a queue to send to the reducer tasks.

4. Reduce Phase

   The JobTracker can launch TaskTracker processes to run the reduce section on the sorted key/value pairs from the map section. The output is sent to the HDFS.

3.5 Comparison with Existing Schemes

There are other systems that are available for automating parallel process management and communication. The systems in question [29] and [30] are very efficient for managing clusters of machines and would be capable of running tasks of Weka broken down into parallel. The main hurdle to these systems is the set-up cost. There are number of different architectures that have built a parallel architecture with Weka. Weka-Parallel [31] is one example which uses Java RMI over multiple hosts to spread the load of validating the algorithms. Another such library is grid enabled Weka [32]. The disadvantage of these libraries is that there is a requirement for dedicated hardware and maintenance of the application across a number of hosts. The architecture described here will be capable of plugging into an existing Hadoop infrastructure.

3.6 Recap

Hadoop is an attractive option as it is an established platform built to handle large loads of data. It is also built with Java which fits nicely with weka. It has an excellent document base. A quick reference to stackoverflow [6] shows 4,338 tagged questions so there is an active community around the product. There are lots of different options on inserting the data into the cluster and methods to scale different sections of the architecture depending on the requirements of the user. Weka is open source and the code is available to examine or change depending on the needs of the application. The flexibility of both platforms will become important in the next chapter as the design and implementation phase will discuss the various options available to meet the needs of this project. The capacity to performance tune the platform will also be vital as the fact that each map process will run in its own Java virtual machine will mean that there are options (briefly discussed in chapter 2) to ensure that it is suited to running a data intensive process.

[6] www.stackoverflow.com

Chapter 4: **Parallel Architecture for Machine Learning**

Chapter 2 provided a background into the thought process behind scalable computing and how machine learning is ideally suited to these types of systems. Using the components described in Chapter 3, this section will provide a full description of the various elements of the parallel architecture. There will be an outline of problems encountered during the design, a breakdown of each component and also a section on tuning the system to allow it to handle the demands that will be placed upon it in Chapter 5. The report is now entering the design section and so a refresh of the aims of the report are outlined below.

4.1 Requirements

- The system will be able to evaluate machine learning data.
- Build a scalable, robust system capable of breaking the machine learning process down into parallel tasks specifically Cross Validation and Parameter Selection for Machine Learning Algorithms
- Provide a full report on the scalability metrics of the system
- Plug and Play Architecture

4.2 Options for Building the System

The first questions that were looked at for a potential parallel architecture are the components that need to be completed in parallel. In terms of Cross Validation, what are the elements that would be best suited to running in the map process and what would run in the reduce process? Parameter Selection is a little more straight forward as this involves running the complete process on the map and then comparing the results on the reduce process.

4.2.1 Experimenting with Weka

Running a simple example of Weka and putting timing details in the code to outline how long the various stages of execution takes will provide a good insight into what parts of the code are taking longest to run. It will also provide details on what are the most intensive parts of the process.

The derived average value from Table 4.1 is 56804 milliseconds, it is clear that the test phase of the process is taking a significant amount of time and CPU. Therefore both the training and testing will be performed on each map in Hadoop and the results will be passed

FOLD										
CV	1	2	3	4	5	6	7	8	9	10
10	56760	58181	56104	55732	55802	56259	56060	57488	56944	58717

Table 4.1: Average Time for Testing a standard set of data in Weka

to the reduce process where the results will be generated. The reduce process will involve taking the results of each map and either deciding the best result (Parameter Selection) or combining the results together (Cross Validation) to provide the final result. All the intensive data processing will take part in the map segment of the architecture. The early test above indicates that the map process could become a system bottleneck.

4.3 Design - Purpose of the model

In order to implement these requirements, Weka will need to be distributed to a series of nodes that will run their section of the machine learning algorithm. The output will then be combined once all the jobs are completed and the output generated to the end user. In terms of Hadoop this means that each map will run its section of data, the results of each map will be combined into the reducer which provide the output and final result. The architecture will be modular in that the user will be able to pass different machine learning algorithms into the system. The various parts are listed below.

4.3.1 Infrastructure Set-up

The Hadoop cluster will be run in pseduo-distributed mode. [1] This essentially means that Hadoop will run on a single host but all the individual components will run as separate Java processes. This will suit the parallel architecture as each component of Hadoop will be launched as a separate JVM process and it will allow the report to measure how the system will scale across varying amounts of CPU cores by assign specific JVM's to CPU's. The appendix contains a full description on how the files were moved into the HDFS.

4.3.2 Inputs

This is a key step in the application. Typical Hadoop applications let Hadoop decide on the number of mappers and the number of blocks that get delivered to each mapper in the system. The way the input file is split up is defined by one of the implementations of the "InputFormat" interface. There are several options which are discussed below.

- TextInputFormat

  This is the default mode for passing data. Each line in the file is a record and the key is the byte offset of the line.

- KeyValueTextInputFormat

  Again each line in the file is a record. A separator character divides each line. Everything before the separator is the key.

[1] http://hadoop.apache.org/docs/r0.20.2/quickstart.html

- NLineInputFormat This has the same functionality as the first option - TextInputFormat. The interesting feature of this option is that it is possible to set the number of lines that will be contained in each split.
- SequenceFileInputFormat $\langle K, V\rangle$

  In this case, the key and value are defined by the user. This mode is especially useful for passing data between multiple map reduce jobs.

Each of the options above is a good option for reading in large datasets where the user is looking to efficiently break down the data into chunks and process each chunk independently of the other sections. In the case of this architecture there are two key items that the options above do not provide. The first option is that Weka needs to load the entire dataset to perform cross validation as all the data is used in the process of training and testing the system. The second key item is the number of map processes that are launched when the system is run. By default, Hadoop takes care of this process for the user (As stated in Chapter 2, features of scalable systems,this is a desirable feature). In order to decide how many map processes to launch for a particular job, Hadoop will look at the split size as specified in the configuration and then divide the data into the number of map processes that it deems necessary to complete the job.

The "NLineInputFormat" looked promising as it would be possible to count the number of lines in the ARFF data file due to be loaded into the system. Once loaded into the map process, Weka would have access to the entire dataset to perform its functions. It does not however allow the user to specify the number of maps that are launched by the process. There is no simple built in option that will provide the functionality that is needed.

Custom InputFormat

Fortunately Hadoop provides a solution. It is possible to build a custom InputFormat. By implementing this format to suit the project requirements, the framework will use the adhere to the key points listed below:

- This format will allow the user to specify the number of maps to launch.
- The entire file will be delivered to each map process so that Weka will be able to process the full ARFF file.

The system will now allow the user to control the number of maps - this means that a separate map for each CV fold can be run or in the case of parameter selection, a map can be run to test each option in trying to evaluate a certain algorithm

4.3.3 Map Process - Integrating Weka

Cross Validation

Now that the data will be delivered to each map, the Weka libraries will be used to start evaluating the data. A quick look at how Weka performs CV.

Please note that this system will divide each CV fold onto a different map process. E.g. 10 Fold CV = 10 map processes.

```
Instances trainInstance = dataSet.trainCV(numberOfFolds, TestSlice);
Instances testInstance = dataSet.testCV(numberOfFolds, TestSlice);
```

To perform Cross Validation in each mapper process, two pieces of information are needed - the number of folds and the slice that will be withheld for evaluation/ testing. The number of folds can be passed in by the user and stored in a global variable (Hadoop allows distribution of global values that can be accessed from inside distributed map processes). Since the slice number will be variable depending on the map/ fold number, a design decision was made to use the Hadoop Job map number. So the process will look like the following:

```
Instances trainInstance = dataSet.trainCV(NumberOfMaps, JobMapNumber);
Instances testInstance = dataSet.testCV(NumberOfMaps, JobMapNumber);
```

The Job Number is accessible from the mapper so Map 1 will be listed as Job 1 and this process will continue up until the number of folds desired by the user is reached. This system will continue to work even in the event of job failures as Hadoop maintains the same job number when it reruns the failed process.

Parameter Selection

The same theory will hold performing parameter selection with the system. Instead of performing a separate cross validation fold on each map, the system will run an entire training and testing set on one map. This process is similar to running a command line version of Weka on multiple mappers, in the case of the nearest neighbour algorithms, the value of K will be different on each map process.

The approach taken with Parameter Selection was to allow the user to enter a range of values. The system would then launch a set of map processes to test each parameter. The input parameters to the system are the range of values and a step size. The system can then decide how many map process to launch from this information.

INPUT	
ALGORITHM	NEAREST NEIGHBOUR IBk
START	2
END	10
STEP	2
SYSTEM DERIVED VALUES	
K VALUES	2,4,6,8,10
NUMBER MAP PROCESSES	5

Table 4.2: Parameter Selection Breakdown

From Table 4.1, the system will train and test the Nearest Neighbour Algorithm using values of (K=2,4,6,8 and 10). This corresponds to five map processes.

4.3.4 Reduce - Combining the Weka Outputs

When the Weka libraries perform CV on the data, the output is stored in an "Evaluation" object [2]. Each mapper process will deliver an evaluation object to the Hadoop reducer which

[2] http://weka.sourceforge.net/doc.dev/weka/classifiers/Evaluation.html

must now iterate through each object adding to an overall result. One problem encountered with this process is that the standard Weka libraries have no built in method of combining separate versions of "Evaluation" objects. Early versions of the system in this report used a custom built object to combine the various map outputs. This approach was inspired by the methods employed by the code used to build Weka parallel [31].

A design decision was made to use the latest branch of the Weka libraries as a new "AggregateEvaluation" object has been added to the code base which suits the needs of this project and provides a built in method of combining "Evaluation" objects together.[3] This new object is the direction that the Weka libraries will take so it is best to stay consistent with future versions.

Listing 4.1: Reducer Sample with Weka.

```
 public void reduce(Text key, Iterable<AggregateableEvaluation> values, Context context){
   for (AggregateableEvaluation val : values)
   {
       if (sum == 0)
       {
          try {
                aggEval = val;
                }
          catch (Exception e) {
                e.printStackTrace();
                }
       }
       else
       {
           aggEval.aggregate(val);
       }
       sum += 1;
   }
}
```

The code sample in listing 4.1 shows a sample of a list of "AggregateEvaluation" being passed from the map to the standalone reduce process. The code sample outlines a method of combining the objects together.

4.4 Tying the System Together

Now that each of the phases of the system have been outlined, another important step is the job set-up. This is essentially the "Main" method of a Hadoop program. It is at this stage that the inputs and outputs of both the map and reduce are defined. This sets the general environment up for the job that is about to be passed through the system. The global variable that can be accessed from each map and reduce process can also be set here.

[3]Response by Mark Hall, core Weka contributor - http://weka.8497.n7.nabble.com/Merge-Cross-Validation-folds-results-td18207.html#a34521800

```
// SET THE GLOBAL VARIABLES
conf.setInt("Run-num.splits", Integer.parseInt(args[0]));
conf.setStrings("Run.classify", args[1]);

// TELL THE SYSTEM WHICH MAP AND REDUCE CLASSES TO USE
job.setMapperClass(WekaMap.class);
job.setReducerClass(WekaReducer.class);

//This sections set the values of the <K2, V2>
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(AggregateableEvaluation.class);

//REMEMBER THE CUSTOM INPUTFORMAT FROM SECTION 4.3.2
//In this case we are overriding TextInputFormat
job.setInputFormatClass(WekaInputFormat.class);
```

Recall Table 3.1 where the Hadoop map reduce format is illustrated, Table 4.3 outlines the process as it applies to this architecture. This would be the classical view of the flow of a MapReduce program.

	INPUT	OUTPUT
MAP	$\langle object, Text\rangle$	list$\langle Text, AggregateEvaluation\rangle$
REDUCE	$\langle Text, list(AggregateEvaluation)\rangle$	list$\langle Text, AggregateEvaluation\rangle$

Table 4.3: Customised Map Reduce outline

4.5 Performance Tuning

There is now a working architecture in place capable of using certain Weka features in a parallel framework. The process now turns to enabling the features of Hadoop that would make it suitable for running data intensive applications. The following list describes some of the main issues encountered when running machine learning algorithms on the Hadoop cluster.

- Map Process running out of memory

  Each map process that launches will use up a certain amount of memory loading in the full ARFF file from HDFS. Weka running a machine learning algorithm to train and test the data will also consume memory. Running the system with the default settings leads to an "Out of Memory" alert from Hadoop during the map phase. To counteract this issue, the amount of memory allocated to each map process is 2 GB.

  ```
  <property>
      <name>mapred.child.java.opts</name>
      <value>-Xmx2048m</value>
  </property>
  ```

- Map Timing Out

  Hadoop includes a default timer on each map process where it will terminate the task if it is not reading an input, writing an output or updating its status. The default time is 600 seconds. Due to the fact that this system is training and testing a body of data, the map process can go over 600 seconds. To stop the process being terminated, the timeout is increased

```
<property>
  <name>mapred.task.timeout</name>
  <value>36000000</value>
</property>
```

- Datanode Heapsize

  When dealing with a small number of files on a small cluster (in this case one host), the datanode memory usage climbs very fast because there are a limited number of places that replication can occur. Adding more hosts to the cluster would solve this problem but since this system is being run on one host, it is necessary to add more memory to the datanode.

  ```
  export HADOOP_DATANODE_OPTS="-Xmx4g"
  ```

- Progress Reporting

  There is a reporting mechanism available that allows the map and reduce processes to report back to the Job Tracker what the current status is. This is a useful feature for debugging where the map/ reduce is spending a lot of its time. This is reported back through the context object that is passed into each map and reduce as part of the Hadoop framework.

  ```
  context.setStatus("Evaluating the model");
  ```

- Optimizing Map output

  Another useful feature is the ability of the map task to sort its output before pushing to the reduce process. During the testing of this architecture, its was noted that increasing the amount of in-memory buffer available to the map task would be more efficient for sorting the output.

  ```
  <property>
    <name>io.sort.mb</name>
    <value>400</value>
  </property>
  ```

- Limiting Hadoop Resources

  As part of Chapter 5, one of the experiments outlines a test involving limiting the resources (CPU) available to the Hadoop framework. It is the Task Tracker component in Hadoop that creates and manages the child processes that perform the heavy lifting in the map and reduce sections. One interesting project [37] outlines how by manipulating the code in the task tracker, it is possible to assign specific CPU cores to a task. For this project, it was found that by changing the start-up scripts for Hadoop specifically the Task Tracker component, it was possible to control which CPU cores the processes ran on. The start-up script in question is called "hadoop-daemon.sh" and is found in all standard Hadoop installations.

  ```
      if [ "$command" == "tasktracker" ]; then
          cd "$HADOOP_PREFIX"
          taskset -c 1-2  nohup nice -n -20 /usr/local/bin/hadoop --config $command
      fi
  ```

The taskset command above will mean when the Task Tracker is started, it will run only on CPU's 1 and 2.

In the Hardware tuning section in Chapter 2, there was a section on the option of setting the nice value. It is worth noting the value of the nice command that is also passed to the process (-20) meaning that this process will be assigned top priority by the operating system.

4.5.1 Accurate Run Times

Another aspect of the system that will become very important in Chapter 5 is the ability to measure the time that each individual component takes to run in the system. In the first instance this is a very useful measurement in order to find potential bottlenecks; it is also the main tool used to compare different instances of the system against each other. By logging a simple time-stamp before and after significant operations (training, testing), it is possible to calculate the time taken to complete each task.

One of the most useful features of Hadoop is its built in web browser. Through this interface, it was possible to output the run times of each job that passed through the system.

4.6 Recap

The parallel architecture satisfied all requirements outlined at the beginning of the chapter. One of the major plus points of Hadoop as a framework is the fact that by deploying the code for Cross Validation and Parameter Selection as external ”jar” files, they can be submitted to the system without any change to the underlying architecture.

The ability to control the number of map processes provides a very flexible architecture. This is a non standard way of using Hadoop in that the system is built towards dealing with massive datasets which can be accessed in blocks and distributed to nodes that sit closest to the data. This shows the true power of the platform in that there are options available for customising the framework for uses that it was not intended for.

During the development of the framework, two problems that were outlined in the chapter - a custom evaluation object and launching a Task Tracker on a specific CPU core. For both these issues a customised solution was followed and implemented but in later versions of the software, alternative methods of achieving those goals were found within the existing framework. From a design point of view, this was good decision as it is important to stay within the frameworks of these tools as much as possible as it means that the work will be accessible to other groups that are also using the platforms.

Chapter 5: **Experiments and Evaluation**

In order to look closely at the performance of the system, the standard techniques for measuring performance in parallel systems is used. The various measurements (Speed-up, scale-up and size-up) have been discussed in Chapter 2. A full description of the environment used for the experiments is provided along with a discussion of the goals and results of each evaluation. Please note that while the experiments involve passing a body of data through machine learning algorithms, the accuracy of the algorithms is not the main goal but rather a discussion on who the system scales to meet the demands of passing this data through in a parallel format.

5.1 Aims

The aim of the experiments is the following:

1. Compare the total running time of weka against the parallel architecture
2. Analyse the effects of limiting computing resources on the parallel architecture
3. Calculate the speed-up, scale-up and size-up provided by the parallel architecture

5.1.1 Lab Environment

This section presents the experimental results for running the system on an Intel multi-core machine. All experiments are carried out on an Intel 32-core machine with 2 Xeon 2.9GHz 8core chips. This makes 16 cores in total and with hyper threading turned on brings the number of available cores to 32. The system is running DDR-3 1600 with 32GB RAM. The make and model of the machine running the tests is a Dell R610 2012 model. The OS release is Suse 11.2 with kernel version 3.0.13-0.27-default. The version of Hadoop used in the testing is hadoop-1.0.3 (the stable release at the time of testing). The version of weka being used in the experiments is 3.7.0. The latest library is being used in order to take advantage of some advanced functionality in merging similar datasets. This was discussed in chapter 4. All input data is located on the HDFS file-system that is on the host itself. The output data is also generated on this host.

5.1.2 Description data-set

The data was taken from the UCI repository [1]. A cut down version of the dataset is used in the experiments.

[1]http://archive.ics.uci.edu/ml/datasets/KDD+Cup+1999+Data

5.1.3 Explanation of Terms

- Classification: In Weka terms this is where the data model is built
- Evaluation: This is where the testing of the model occurs
- Standard Weka: Running Weka in command line mode.

5.2 Benchmark Weka against Parallel Framework

The purpose of this experiment is to demonstrate the speed-up provided by the parallel architecture over standard Weka. Cross Validation folds from 2 to 10 will be generated on both the stand alone Weka binaries and the parallel architecture. Results will be illustrated using both the J48 and Naive Bayes machine learning algorithms. There will be a full comparison of the time to run each algorithm through each cross validation fold.

5.2.1 Test Set-up

- Run Weka on the command line with each CV fold numbered 2 to 10

  ```
  java -cp weka.jar -Xmx2048M weka,classifier.trees.J48 -t inputFile -x 2
  ```

- The test harness above will automatically calculate times for each of the cross validation folds from 2 to 10

  The harness will also system details to a log, the length of each fold will be written to csv file format. A full description of the test harness is available in the appendix

5.2.2 Machine Learning Algorithms: J48 Tree & Naive Bayes

Figure 5.1 contains the run times for the machine learning algorithms J48 and Naive Bayes. The first detail that can be derived from the graph is that there is a definite speed-up when comparing the parallel architecture to Weka on both machine learning algorithms. The Weka software is running in a sequential mode. Each cross validation is carried out in a loop and added to an overall result. The parallel framework however calculates each cross validation fold in parallel. In the case of using the system with Hadoop, each fold corresponds to a mapper process in the architecture. Some key results are outlined below:

1. The time for Weka to calculate 10 fold CV was 5263 seconds, the equivalent time on the parallel system was 689 seconds for the J48 algorithm
2. For Naive Bayes, Weka time for 10 fold CV was 1279 seconds, parallel system was 175 seconds.
3. The Naive Bayes output is operating as expected. The time for running Weka is increasing with the number of folds while it is heading in a downward direction in the parallel framework

   The same is not happening with the J48 algorithm however. The time to compute Weka J48 is heading upwards (which is expected) but the time to calculate parallel J48 is also sloping slightly upwards. Further analysis will be carried out in the section below.

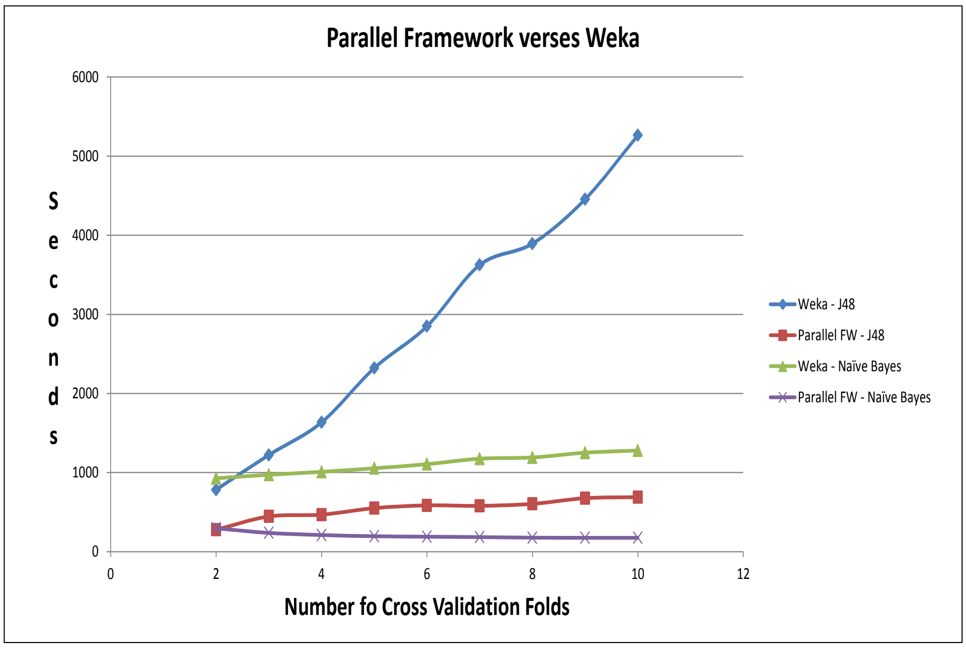

Figure 5.1: Benchmark on Weka verses Parallel Architecture.

5.2.3 Discussion

Fig 5.2 and Fig 5.3 give a further breakdown of the iterations of the parallel architecture. The evaluation time is moving downwards as the number of folds increases (The amount of data to evaluate would decrease with an increase in the number of folds). The classification time is moving upwards as the amount of data that needs to be classified is increasing with each fold increase. The amount of time each process spends classifying the data is increasing as the number of folds increases, this is increasing more than the time spent evaluating the data. This is the reason that the parallel plot in figure5.1 is increasing.

Naive Bayes behaves in a slightly different manner. The Weka benchmark is getting larger (the more cross validation folds the more time it takes) while in the parallel framework, the graph is heading downward. The reason for this occurrence is that the time to test the data (evaluate) is at its largest with two folds. As more folds are added to the cross validation, the amount of data to test is getting smaller.

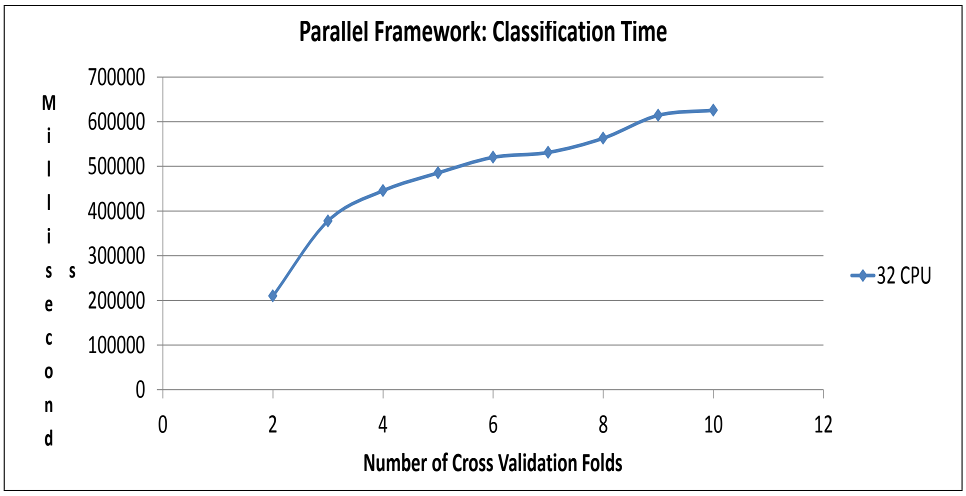

Figure 5.2: Training times for parallel architecture - J48.

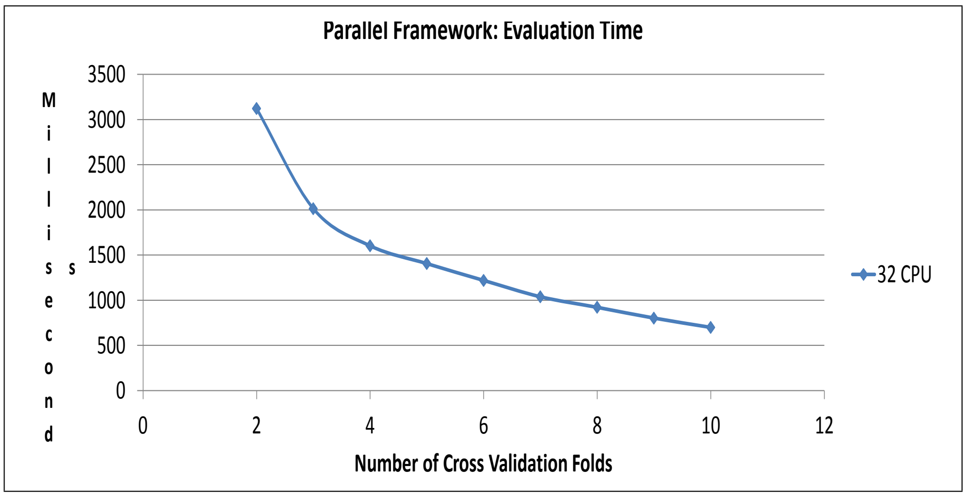

Figure 5.3: Testing time for parallel architecture - J48.

5.3 Parallel Framework Limited Resources Performance

The results from experiment 1 show that the parallel architecture is faster when using the entire resources available to the system. (32 processors in this case). The experiment below outlines a breakdown of limiting the resources of the system to certain CPU's. Fig 5.4 outlines the run times of the parallel architecture when limiting CPU usage from 1..10 processors. The max usage of 32 processors is also included in the output. The results from the J48 algorithm are outlined in the section below.

Aim

- In this experiment the system will run 11 times, each run includes generating CV folds from 2..10.
- Each run will have access to limited CPU resources. First run will have 1 CPU, Second run will have 2 CPU's all the way up to 10 CPU's

Test Set-up

- System resources will be limited by using the "taskset" configuration outlined in Chapter 4.
- The test harness will automatically calculate times for each of the cross validation folds from 2 to 10. The harness will also system details to a log, the length of each fold will be written to csv file.

Results

The graph in Figure 5.4 illustrates the difference in running the system over a variable number of CPU processors.

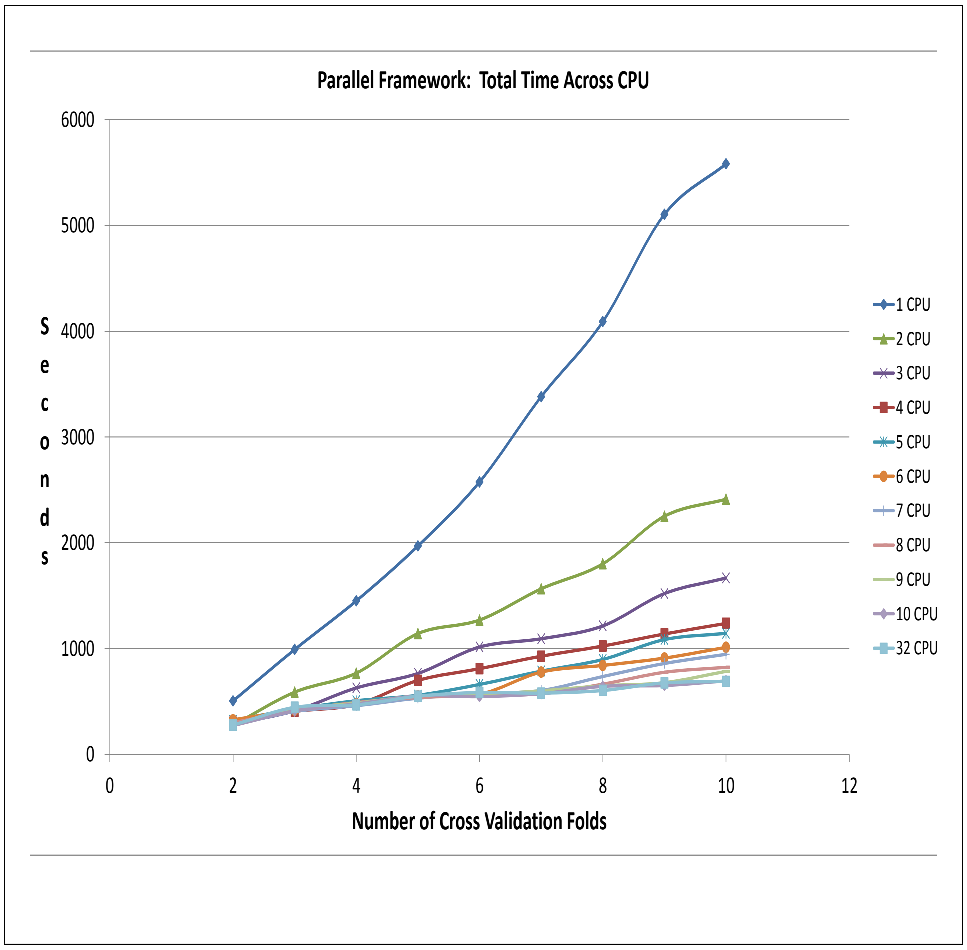

Figure 5.4: Total CPU time for parallel architecture.

5.3.1 Discussion

It should be noted that the performance of the system is at its worst when limiting the application to one cpu. As the number of folds increases the run time is increasing in a linear fashion. As the number of processors available to run each map increases the performance of the system is also improving. The improvements become smaller and smaller with small incremental increases from processor 7 onwards. It should be noted that the best 10 fold cv result was still achieved by the utilisation of the full 32 processors. Another important point is that when running these tests, it is only the map and reduce processes that are operating off limited CPU resources. The remaining Hadoop components (JobTracker, Data Note, Name Node and Secondary Name Node) are all operating the remaining CPU's available to the system.

5.4 Measuring the Mapper and Reducer Performance

This section will examine the performance of the map and reduce section of Hadoop. It is this architecture that allows the system to break down tasks in parallel. The two key figures (5.5 and 5.6) outline the performance of both the mapper and the reducer. The key points to note are outlined below

5.4.1 Mapper

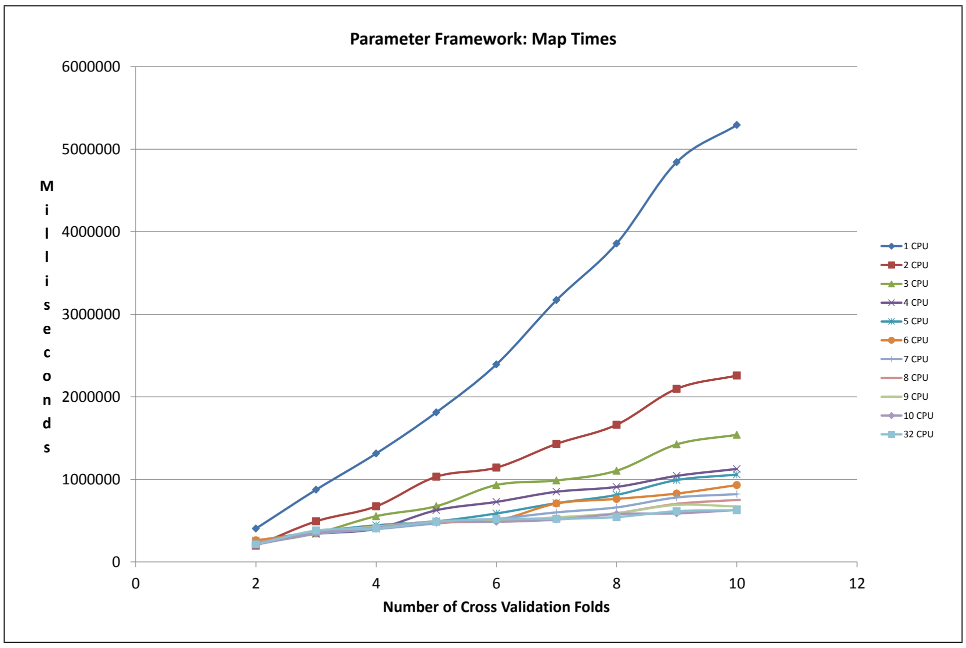

Figure 5.5: Map Performance.

- The performance of the mapper is largely in line with the performance of the training and the testing numbers in earlier figures.
- There is a large performance penalty when using one CPU for this type of work
- While initially adding cores to the framework yields some impressive performance increases, the benefits decrease as more cores are added to the system.

The fact that the performance of the map process compares favourably to the performance of the elements of the machine learning algorithms that are running in this section of the system (In fact in this architecture, all training and testing is carried out on the map process) means that there is no significant overhead being added by the process of starting and stopping the map framework. Again from the graph, there is a marked performance improvement when moving to more than 1 CPU.

5.4.2 Reducer

Given that the data intensive tasks are occurring in the map section of the architecture, even though the reducer is a sequential operation, there is a difference between running the process on multiple cores and on one core. Comparing the results of the reduce process running on 1 core, 10 cores and 32 cores are illustrated in Figure 5.6.

- The reducer will need to wait for all mapper processes to complete before it is allowed to process its jobs. This can add to the overhead.
- In Fig5.6, the speed-up provided by multi-core is not as pronounced. Running the system on one core is still the slowest method but the gap is significantly smaller than in the map phase.

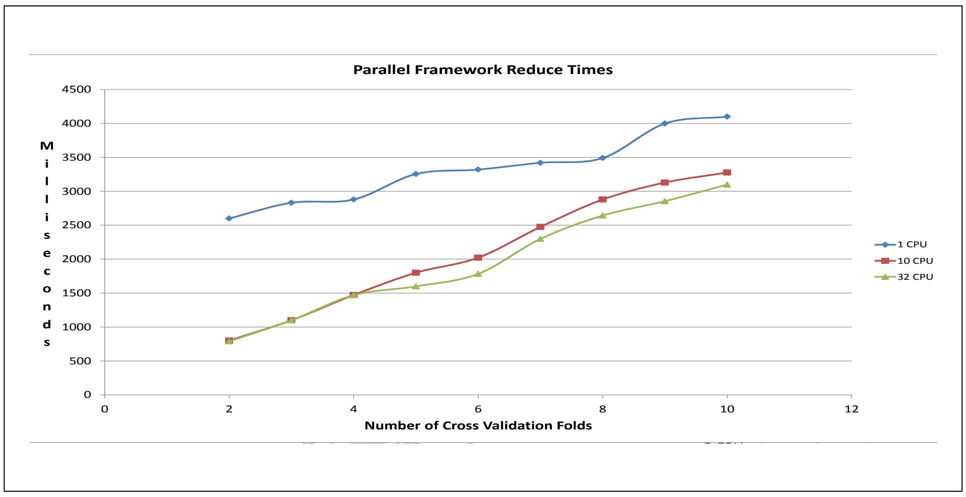

Figure 5.6: Reduce Performance

5.4.3 Discussion

The reducer running on one core should not impact the performance of the system as much as the map phase as it is a sequential operation with only one active process at any one time. The main bottleneck that could be caused by the reducer would be the entry point as up to 10 CV process feed the results into one process. Due to this limitation, it is important that the transition from map phase to reduce phase is as efficient as possible hence the focus on the tuning parameters discussed in Chapter 4.

5.5 Measuring Scale

In the previous sections, there was an outline and discussion of the results of running Weka in sequence and the speed increases provided by the parallel architecture. The supplied graphs proved that the parallel architecture is achieving significant results in training and testing the data. The following section will examine the benefits that are gained by using this parallel architecture by using some common parallel architecture tools for measuring performance. There was a previous discussion on these items during Chapter 2.

5.5.1 Speed-up

Speed-up is defined as the ability of the parallel architecture to improve the running time of the system. It is measured as follows:

$$Speedup(m) = \frac{T(1)}{(T(m)}$$

Applying this metric to the data in Figure 5.4 (Total CPU time across parallel process) yields the table 5.1. Plotting this data in Fig 5.7 shows how the system is scaling as more and more CPU's are added. There is an almost linear improvement when comparing the results of running 1 CPU and 32 CPU's. It is interesting to note that the performance achieved by 10 CPU's is comparable to 32 CPU's. It is clear that the ability to allocate a complete CPU to each cross validation fold generates the best performance gain. Another point of note in Fig 5.7 is how the speedup metric flattens on the graph as the number of cores is greater than or equal to the number of cross validation folds. The CPU intensive nature of the machine learning algorithms is suited to the parallel framework.

	CPU										
CV	1	2	3	4	5	6	7	8	9	10	32
2	1	1.57	1.75	1.67	1.57	1.54	1.84	1.69	1.86	1.86	1.82
3	1	1.64	2.01	2.29	2.32	2.38	2.35	2.39	2.45	2.55	2.67
4	1	1.71	2.41	2.78	2.86	3.04	3.14	3.11	3.29	3.33	3.43
5	1	1.78	2.58	3.00	3.32	3.67	3.71	3.82	3.57	3.92	4.18
6	1	1.96	2.74	3.31	3.83	4.23	4.39	4.63	4.67	4.72	5.10
7	1	2.07	3.09	3.64	4.14	4.56	5.01	5.30	5.63	5.69	5.84
8	1	2.17	3.37	3.99	4.56	4.96	5.47	6.00	6.31	6.36	6.77
9	1	2.27	3.50	4.49	4.90	5.31	5.94	6.58	7.13	7.34	7.55
10	1	2.31	3.66	4.81	5.30	5.82	6.30	6.86	7.56	8.03	8.30

Table 5.1: Speed Up Metric

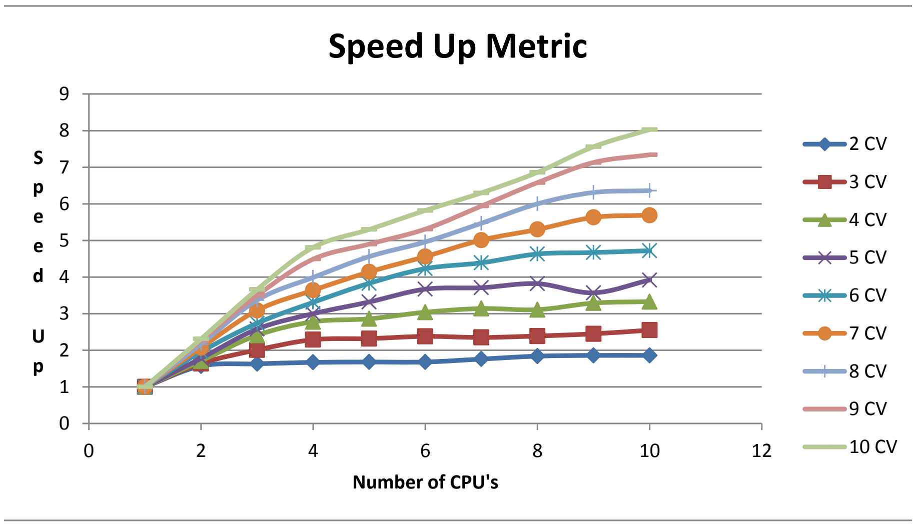

Figure 5.7: Speed Up Metric Performance

5.5.2 Scale-up

Scale-up is defined as measuring the growth of both the system and the data size. In this case, it could be number of CPU's and increase in data size. The example outlined here calculate the execution time of 1 CPU with data size D and then increases to m computing nodes with data size m times of D.

$$Scaleup(m) = \frac{T(1, D)}{(T(m, mD)}$$

The scale-up of the system is illustrated in Figure 5.8 used the derived values from Table 5.2. Calculating 10 fold CV shows that the system scales well going from two cores up to ten cores without a severe drop in the scaleup value. The results prove that the system is able to handle growing data rates.

CPU										
CV	1	2	3	4	5	6	7	8	9	10
10	1	0.53	0.50	0.49	0.48	0.46	0.45	0.44	0.42	0.40

Table 5.2: Scale Up Metric Data

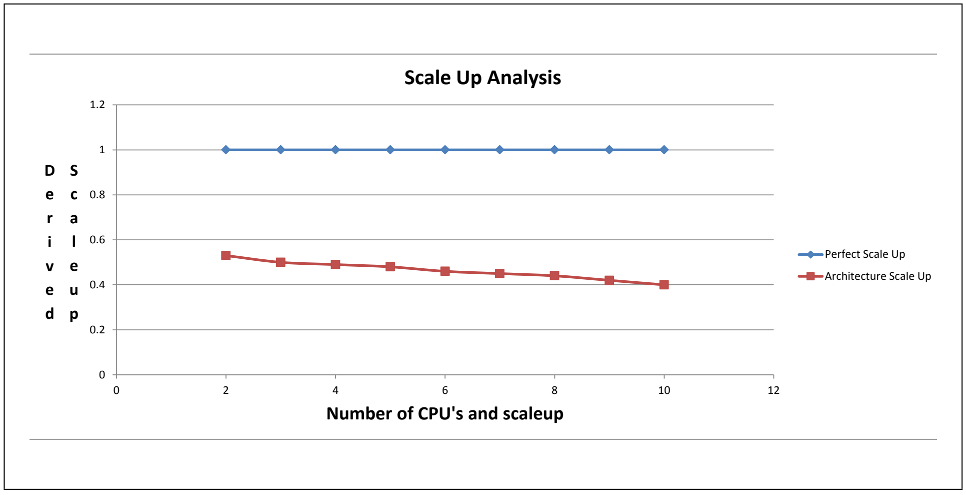

Figure 5.8: Scale Up Metric Performance

5.5.3 Size-up

Size-up is defined as the ability of the system to handle data growth. In this case the amount of nodes used in testing will remain constant and the dataset will grow.

$$Sizeup(m,n) = \frac{T(m,nD)}{(T(m,D)}$$

Data Size										
CPU	1	2D	3D	4D	5D	6D	7D	8D	9D	10D
1	1	3.57	4.55	5.39	6.58	7.42	9.68	10.49	15.28	16.71
2	1	3.13	3.79	4.50	5.03	6.10	7.55	8.56	11.74	13.43
4	1	3.36	3.60	4.41	5.31	5.64	7.29	7.94	10.54	12.37
6	1	2.83	3.44	3.96	4.75	5.43	7.03	7.10	10.08	11.54
8	1	3.11	3.64	3.87	4.80	5.70	7.26	7.57	10.04	11.22
10	1	2.83	3.27	3.57	3.99	4.99	5.69	6.26	8.61	9.64

Table 5.3: Size Up Metric Data

Figure 5.9 shows the results of calculating the size up metric against the data in Table 5.3. Each line corresponds to a certain number of processors used with the framework. Each line shows that the system is able to handle the increases in data size. The key to this metric is examing how much longer the system takes to execute against the data when the data is increasing in size. In this case, when the system is run with 10 processors at 10 times the original data size, the sizeup is 9.64. This points to the system being able handle the data growth.

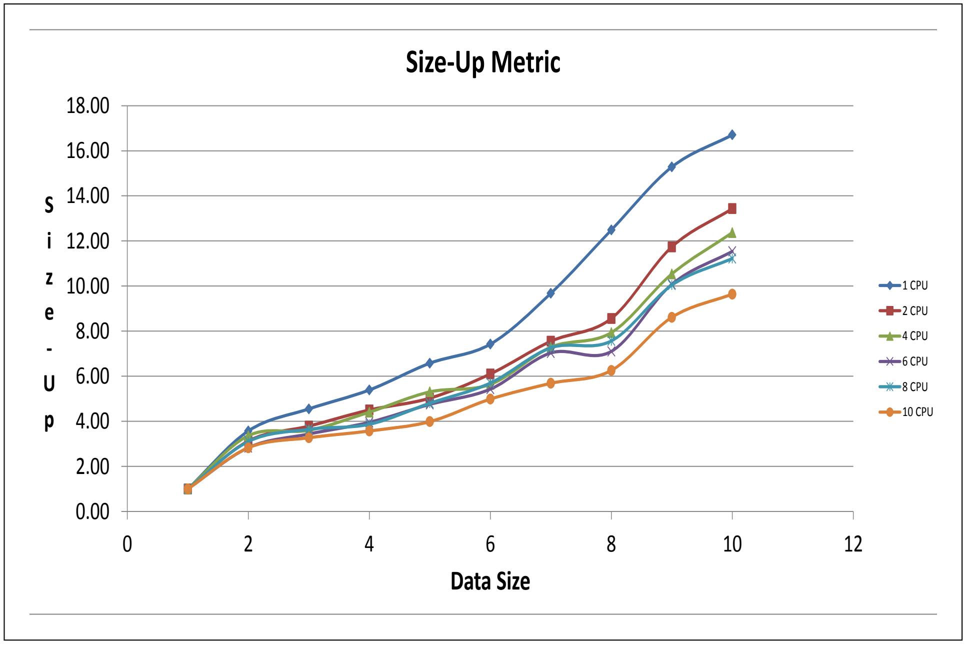

Figure 5.9: Size Up Metric Performance

Chapter 6: **Conclusions**

6.1 Project Recap

In this project a method of validating data in machine learning algorithms was presented. The Weka platform was used for applying the algorithms and using the open source code from this platform to produce a parallel model for the processes of cross validation and parameter selection in machine learning. While Weka could be described as the engine of the system with its built-in support for the ML algorithms, Hadoop could be described as providing the infrastructure for the project. It provided the scalable and flexible platform and defined the inputs and outputs to the architecture. One of the major strengths of Hadoop is its ability to allow a functional programming idea such as MapReduce and allow an object oriented language such as Java to use its functionality to solve data intensive problems.

The implementation was unique in that it combined the features of Weka and Hadoop pushing data in a specific file format (ARFF), overriding the input parameters and delivering the data to each map process. There are some projects that attempted to build a parallel Weka but none used the same combination of technologies and parameters. During the research of this project, each of the aforementioned projects was examined to see if any lessons could be learned in terms of building the system to provide the largest scale-up possible. [33][34][35][36]

The goals of this project were to build a MapReduce framework for evaluating machine learning algorithms and this was achieved but another important goal of the project was to examine how scalable the platform would be. This report outlines several scalable metrics including speedup, scaleup and sizeup to define the metrics of the parallel architecture.

6.2 Conclusions/ Results

The results of the project are encouraging, there is a marked speed-up as more CPU cores are added to the platform. The parallel metrics provide results with positive metrics present in the graphs outlining the achieved speed-up, scale-up of the system and the size-up as the data increases in size. The findings confirm that it is possible to achieve some significant savings on data intensive tasks by moving them to a parallel architecture. For example, the speed-up metric generated by 10 fold cross validation on 2 cores is 2.31, 10 cores is 8.03 while on 32 cores the metric is 8.30. An important inference from these results is the benefit in assigning at least a single core to each fold in cross validation.

The MapReduce model proves to be very useful and the framework provided by Hadoop is very flexible with lots of input and output options. There has been a lot of work on making the system as scalable as possible as seen in Chapter 4 when discussing options to tune the system.

Although the aim of the project was to build a MapReduce framework and measure the scalability of the platform, another important point to consider is that the results of running each algorithm and the results it generates were carefully considered during the design and implementation of the framework to the point that the generated results were compared against the results of running the same data through a standalone version of Weka.

A possible limiting factor with the architecture is the amount of physical memory available on the host running the architecture. In this project, the aim was to run the platform on one host and measure the scalability by using the amount of cores available. One observation was as the data size increased, the Weka libraries need to load the entire dataset into memory. As the system breaks the process down in parallel tasks (10 Fold Cross Validation), the data is loaded into memory in conjunction with the number of parallel tasks launched. This can lead to memory issues that no amount of tuning can alleviate as there is a sheer lack of physical memory to run the framework. Given that the system is built against Hadoop, it is very easy to extend the cluster to more than one node and this would remove the physical memory obstacle.

6.3 Possible Future Work

While the system demonstrated performance gains by implementing MapReduce and Weka, there some more options that would be worth pursuing to extract extra performance from the system.

- Build a more efficient Scheduler for Hadoop TaskManager

  There is an interesting improvement outlined [37] [38] on how the TaskManager in Hadoop spawns its map task for process in the map stage of the system. The curent method used by Hadoop is that this is handled by the TaskManager which simply spawns a new jvm for each mapper. If the system was made aware of the processor architecture of the host, an interesting angle would be to have the TaskManager assign specific CPU cores to certain tasks. In this case the system would effectively become NUMA aware and would run the architecture in such a way as to make the best use of memory resources available to the processor.

- Load the Weka data to the Map process in a more efficient manner

  The system described in this report enables the same file to be loaded to multiple map processes. An improvement could be made which the map process could load the ARFF attributes separately from the data.

- Use an alternative MapReduce framework

  The MapReduce programming model worked well to build and scale a parallel system. An interesting project might be see how an alternative system scales against the Hadoop and Weka frameworks. An appropriate model might be the DisCo computing framework [1]

[1] http://discoproject.org

Chapter 7: **Appendix**

7.1 Test Harness

```
#!/bin/bash

now=$(date +"%m%d%y.%H%M")
classifier="weka.classifiers.trees.J48"
inputFile="/home/hduser/input/kddcup.arff"
resultFile="result.txt".$now
analysisFile="analysis.csv".$now
mem="8192M"

topFile="top".$now

top -u hduser -c -b >> $topFile &

echo "START OF TEST HARNESS" > $resultFile
for var in {2..10}
do
    echo "---------------- FOLD STARTING --------------" >> $resultFile

    echo "FOLD: " $var >> $resultFile
    date1=$(date +"%s")
    java -cp weka.jar -Xmx$mem $classifier -t $inputFile -x $var >> $resultFile
    date2=$(date +"%s")
    diff=$(($date2-$date1))
    echo $var, $diff >> $analysisFile
    echo "$(($diff / 60)) minutes and $(($diff % 60)) secs elapsed." >> $resultFile

    echo "---------------- FOLD ENDING   --------------" >> $resultFile

done

pkill -9 top
```

7.2 HDFS Data

```
#!/bin/bash
bin/hadoop dfs -rm /hostname/hadoop/tmp/location/location.txt
bin/hadoop dfs -copyFromLocal /hostname/home/hduser/input/location.txt
/hostname/hadoop/tmp/location/
```

7.3 Operating System Details

```
cat /etc/SuSE-release
SUSE Linux Enterprise Server 11 (x86_64)
VERSION = 11
PATCHLEVEL = 2
```

7.4 CPU Specification

```
cat /proc/cpuinfo | grep processor | wc -l
32

cat /proc/cpuinfo
cpu family      : 6
model           : 45
model name      : Intel(R) Xeon(R) CPU E5-2690 0 @ 2.90GHz
stepping        : 7
cpu MHz         : 2899.926
cache size      : 20480 KB
physical id     : 0
siblings        : 16
core id         : 0
cpu cores       : 8
```

Bibliography

[1] I.H. Witten and E. Frank and Mark Hall *Data mining: practical machine learning tools and techniques* ISBN-13: 978-0123748560

[2] Miroslav Kubat, Robert C. Holte, Stan Matwin *Machine Learning for the Detection of Oil Spills in Satellite Radar Images.* Machine LearningFebruary 1998, Volume 30, Issue 2-3, pp 195-215

[3] L. Page, S. Brin, R. Motwani, and T. Winograd. *The pagerank citation ranking: Bringing order to the web.* Technical report, Stanford Digital Library Technologies Project, 1998.

[4] Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, Dan Steinberg *Top 10 algorithms in data mining* Knowledge and Information Systems, January 2008, Volume 14

[5] Holte *Very Simple Classification Rules Perform Well on Most Commonly Used Datasets* Machine Learning April 1993, Volume 11, Issue 1

[6] Randal E. Byrant *Data-Intensive Scalable Computing for Scientific Applications.* Computing in Science & Engineering 2011

[7] Cullen Schaffer *Selecting a classification method by cross-validation.* Machine Learning Volume 13, issue 1

[8] Ron Kohavi, *A study of cross-validation and bootstrap for accuracy estimation and model selection.* International Joint Conference on Artificial Intelligence, 1995

[9] Robert Kohavi: *Wrappers for Performance Enhancement and Oblivious Decision Graphs.* PhD Thesis. Department of Computer Science, Stanford University.

[10] Robert Kohavi, George John: *Automatic Parameter Selection by Minimizing Estimated Error* Machine Learning: Proceedings of the 12th International Conference, San Francisco, CA

[11] Luke, E.A.: *Defining and Measuring Scalability* Scalable Parallel Libraries Conference, 1993

[12] J. Dean and S. Ghemawat *Mapreduce: Simplified data processing on large clusters* ACM Commun., vol. 51, Jan. 2008, pp. 107-113.

[13] *MapReduce for Data Intensive Scientific Analyses* Proc. 4th Intl Conf. eScience, IEEE Press, 2008

[14] Apache Software Foundation. *http://hadoop.apache.org*

[15] *Parallel data intensive computing in scientific and commercial applications.* Parallel Computing, Vol 280

[16] Jonathan Koomey *Worldwide electricity used in data centers.* Environmental Research Letters, Volume 3, Number 3

[17] Gorton, Greenfield, Szalay:2008 *Data Intensive Computing in 21st Century* Computer Journal Volume: 41 , Issue: 4 Page(s): 30 - 32

[18] Fayyad, Piatetsky-Shapiro:1996 *The KDD process for extracting useful knowledge from volumes of data* Communications of the ACM Volume 39 Issue 11, Nov. 1996 Pages 27 34

[19] Smith, Williams *Performance Solution, Connie U.Smith, Lloyd G. Williams*

[20] G. Moore *Cramming more components onto integrated circuits* Electronics Magazine, Volume 38, Number 8

[21] Pam Frost Gorder *Multicore Processors for Science and Engineering* Computing Science and Engineering, March-April 2007 Volume 9, issue 2

[22] Herb Sutter and James Larus *Software and the concurrency revolution* Queue, 3(7):5462, 2005.

[23] Dean, J., Ghemawat, S. *MapReduce: Simplified data processing on large clusters* Sixth Symposium on Operating Systems Design and Implementation, pp. 137149, San Francisco, CA, USA (2004)

[24] C.T. Chu, S.K. Kim, Y.A. Lin, Y.Y. Yu, G. Bradski, A.Y. Ng, and K. Olukotun. *Map-reduce for machine learning on multicore.* In Advances in Neural Information Processing Systems: Proceedings of the 2006 Conference, page 281. MIT Press, 2007.

[25] Chen, Schlosser: Intel Research Pittsburgh. *Map-Reduce Meets Wider Varieties of Applications* http://www.cs.cmu.edu/ chensm/papers/IRP-TR-08-05.pdf

[26] Markus Klems, Jens Nimis, Stefan Tai *Do Clouds Compute? A Framework for Estimating the Value of Cloud Computing* Lecture Notes in Business Information Processing Volume 22, 2009, pp 110-123

[27] Ananth Grama, et. al. *Introduction to Parallel Computing* Pearson Education Ltd., Second Edition, 2004

[28] S. Ghemawat, H. Gobioff and S. Leung *The Google File System.* Proc. 19th ACM Symposium on Operating Systems Principles (2003).

[29] Becker, Sterling, Savarese, Dorband, Ranawake, Packer *http://www.phy.duke.edu/ rgb/brahma/Resources/beowulf/papers/ICPP95/icpp95.html* ICPP 1995

[30] Douglas Thain, Todd Tannenbaum, Miron Livny *Distributed computing in practice: the Condor experience* Special Issue: Grids and Web Services for e-Science, Volume 17, Issue 2-4, pages 323356, February - April 2005

[31] Sebastian Celis and David R Musicant *Weka-Parallel: Machine Learning in Parallel* Computer Science Technical Report 2002b, Carleton College (2002)

[32] Domenico Talia, Paolo Trunfio, Oreste Verta *Weka4WS: A WSRF-Enabled Weka Toolkit for Distributed Data Mining on Grids.* In Proc. of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases PKDD 2005,

[33] D. Papadimitriou. *DisCo: Distributed Co-clustering with Map-Reduce: A Case Study towards Petabyte-Scale End-to-End Mining* Data Mining, 2008. ICDM '08. Eighth IEEE International Conference

[34] Hwee Yong Ong, MSc Computer Science*Accelerating Data-Intensive Applications Using MapReduce.* School of Informatics, The University of Edinburugh. 2010. http://www.inf.ed.ac.uk/publications/thesis/online/IM100765.pdf

[35] Deyaa Adranale. *Parallelization of weka the data mining toolkit using hadoop.* Master's thesis, Otto-von-Guericke University of Magdeburg, December 2008.

[36] Wegener, D. *Toolkit-Based High-Performance Data Mining of Large Data on MapReduce Clusters.* 2009 IEEE International Conference on Data Mining Workshops

[37] *A Hierarchical Approach to Maximizing MapReduce Effciency* PACT '11 Proceedings of the 2011 International Conference on Parallel Architectures and Compilation Techniques

[38] Rong Chen, Haibo Chen, and Binyu Zang *Tiled-MapReduce: Optimizing Resource Usages of Data-parallel Applications on Multicore with Tiling* PACT 10, September 1115, 2010, Vienna, Austria.

[39] D. Borthakur. *The hadoop distributed file system: Architecture and design.* Hadoop Project Website, 2008

[40] Juan Diego Rodriguez, Aritz Perez, and Jose Antonio Lozano *Sensitivity Analysis of k-Fold Cross Validation in Prediction Error Estimation* IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 32, Issue 3, 2010

[41] Peter Harrington *Machine Learning in Action.* ISBN: 9781617290183