University of Sydney

Master's Thesis

Rolling Window Time Series Prediction using MapReduce

Author:
Lei Li

Supervisor:
A/Prof Philip Leong

A thesis submitted in fulfilment of the requirements for the degree of Master of Philosophy

in the

School of Electrical and Information Engineering
at the University of Sydney

September 2014

Declaration of Authorship

I, LEI LI, declare that this thesis titled, "Rolling Window Time Series Prediction using MapReduce" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at University of Sydney.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ______________________________

Date: ______________________________

“The time of life is short; to spend that shortness basely, it would be too long.”

William Shakespeare

UNIVERSITY OF SYDNEY

Abstract

Faculty of Engineering

School of Electrical and Information Engineering

Master of Philosophy

Rolling Window Time Series Prediction Using MapReduce

by Lei LI

Prediction of time series data is an important application in many domains. Despite their inherent advantages, traditional databases and MapReduce methodology are not ideally suited for this type of processing due to dependencies introduced by the sequential nature of time series. In this thesis a novel framework is presented to facilitate retrieval and rolling window prediction of irregularly sampled large-scale time series data. By introducing a new index pool data structure, processing of time series can be efficiently parallelised. The proposed framework is implemented in R programming environment and utilises Hadoop to support parallelisation and fault tolerance. A systematic multi-predictor selection model is designed and applied, in order to choose the best-fit algorithm for different circumstances. Additionally, the boosting method is deployed as a post-processing to further optimise the predictive results. Experimental results on a cloud-based platform indicate that the proposed framework scales linearly up to 32-nodes, and performs efficiently with a relatively optimised prediction.

Acknowledgements

First and foremost, I would like to express my gratitude to my supervisor, A/Prof. Philip Leong for his guidance, inspiring ideas, encouragement, precious time in reviewing my work and always expecting high standards.

I am thankful to Dr. Richard Davis for his valuable comments and discussions, as well as for reviewing my manuscripts which helped me to produce a well organised thesis.

I am extremely grateful to my colleagues Farzad Noorian and Duncan Moss for their generous supports which helped me to complete this work in a timely manner.

Finally I would like thank my parents and Bo Gao for their love and care.

Contents

List of Figures

List of Tables

To mum and dad

Chapter 1

Introduction

1.1 Motivation and Aims

A time series is a record of variables across time, usually measured at equally spaced time intervals. Time series analysis forms the basis for a wide range of applications including physics, climate research, physiology, medical diagnostics, computational finance and economics [1, 2]. Prediction, in particular, is an important aspect of time series analysis. It can be thought of as a form of data mining, namely forecasting future values based on the analysis of data's historical behaviours.

Historically time series prediction was performed by statistician and analysts. With rapid growth in the number and size of time series, manual inspection of time series has become time-consuming, cumbersome and costly, creating a demand for an automatic system to forecast large number of univariate time series. For example, it is common to have over one thousand product lines that need forecasting at least monthly in a businesses. One of the recent attempts to address this need is the *forecast* package by Hyndman et al. [3]. It includes variants of the most popular automatic forecasting algorithms, most of which are based on as either exponential smoothing or autoregressive integrated moving average (ARIMA) models [4]. This thesis was initially motivated to improve the speed of forecasting using this package. It must be also noted that this package is implemented in R [5], a free software programming language for statistical computing analysis which has become the de facto tool in machine learning and time series analysis.

The past decade has seen tremendous advances in application of parallel computing to various fields. New principles and standards are being created to address different requirements, and algorithms undergo many changes to become scalable. This requires

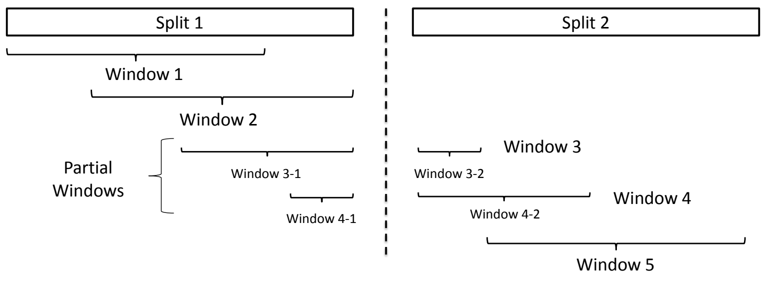


FIGURE 1.1: Issue of partial windows: a rolling window needs data from both windows at split boundaries.

not only an understanding of these algorithms, but of principles and techniques for parallel programming.

To achieve an efficient approach for analysing time series data in a parallel architecture, Hadoop is currently considered as the most appropriate option to try. Apache Hadoop, originally derived from the work of Google's MapReduce [6], has become the standard way to address Big Data problems. MapReduce is used to process files on each node simultaneously and then aggregate their outputs to generate the final result. Hadoop allows for more scalable, cost effective, flexible and fault tolerant parallel programming [7].

Despite all of its advantages, the original MapReduce methodology of Hadoop is not ideally suited for time series analysis. This is due to the implicit dependencies among time series data observations [8]. Therefore partitioning and processing of time series data using Hadoop require additional considerations:

- Time series prediction algorithms operate on *rolling windows*, where a window of consecutive observations is used to predict the future samples. This fixed length window moves from the beginning of the data to the end of it. But in Hadoop, when the rolling window straddles two files, data from both are required to form a window and hence make a prediction (see Figure 1.1).
- File sizes can vary depending on the number of time samples in each file.
- The best algorithm for performing prediction depends on the data and a considerable amount of expertise is required to design and configure a good predictor.

In addition, the issue of predictor algorithm selection and optimisation is critical, as is the implementation of an efficient platform that scales with time series data size.

The main aim of this thesis is to develop a novel framework that can achieve parallel rolling time series prediction using Hadoop. By implementing the proposed framework, the system should be able to deal with massive amount of time series data, either regular or irregular. Furthermore, the proposed system can handle the optimisation, parameter selection and also model combination through boosting.

1.2 Assumptions

In order to clarify the objectives of the thesis, certain assumptions are made in advance to give a better idea of what the proposed system is going to do and why it is significantly important to the time series prediction:

- The time series data is sufficiently large such that distributed processing is required to produce results in a timely manner.
- Time series prediction algorithms have high computational complexity so that an efficient and fast prediction model is urgently needed.
- Disk space concerns preclude making multiple copies of the data.
- The time series are organised in multiple files, where the file name is used to specify an ordering of its data. For example, daily data might be arranged with the date as its file name.
- In general, the data consists of vectors of fixed dimension which can be irregularly spaced in time.

1.3 Contribution

The first contribution of this thesis is a framework for rolling time series analysis using Hadoop [9]. The notion of a new index pool data structure is introduced, which is pivotal for the entire framework to successfully solve the issues of dispatching regularly or irregularly sampled time series data to each computational node. The problems associated with locating the index of time series data, architecture design issues, framework efficiency and flexibility are studied in detail. An efficient architecture is designed which enables the elegant handling of rolling time series forecasting smoothly, by using the MapReduce programming model.

Within the framework, a systematic approach to time series prediction is developed which facilitates the implementation and comparison of time series prediction algorithms in a wrapper model. A user-customisable multi-predictor model (MPM) is comprised of the commonly used predictor algorithms. Applying the MPM in the proposed framework not only allows algorithm auto-selection for a range of different circumstances, but also avoids common pitfalls such as over-fitting and peeking of data.

The third contribution is a feasibility study of deploying the proposed parallel rolling time series prediction system on a cloud-based platform. Considering its scalable processing capacity, cloud computing is a good alternative for performing big data analysis. Furthermore, the MapReduce model is in a suitable form for the applications deployed across cloud clusters. The further evaluation on the architecture performance and enhancement of the scalability are achieved by implementing the proposed framework on Amazon Web Service (AWS) cloud clusters.

The last contribution is a study of applying boosting techniques to the proposed framework for a further prediction optimisation. A rolling procedure is employed within the boosting experiments to enhance the stability and predictive accuracy.

1.4 Thesis Structure

The remainder of the thesis is organised as follows. Chapter 2 gives an introduction to relevant background on time series prediction, MapReduce, Amazon Web Service, boosting and reviews the related work in this field. Chapter 3 proposes the core methodology of using MapReduce to achieve rolling window prediction and the notion of a new data storage indexing design. Chapter 4 compares two popular cloud services and has a further study on AWS cloud service. This chapter also describes the details about how Rhipe package performs in parallel processing. The following Chapter 5 contains the details on the ensemble scheme theory of boosting and gradient boosting machine. The descriptive parameter specifications and implementation of the *gbm* package are included in the same chapter. Chapter 6 presents the experimental results with rigorous analysis and discussion. Finally, the research conclusions are summarised and future research directions are outlined in Chapter 7. Some less important details about boosting techniques are included in Appendix A.

Chapter 2

Background

This chapter establishes the theoretical foundations on which the research in this thesis is based. Specifically, the areas covered are: time series prediction and techniques, rolling time series processing, MapReduce, Amazon Cloud Service and boosting techniques. The work presented in this thesis is an amalgamation of these research fields (see Figure 2.1).

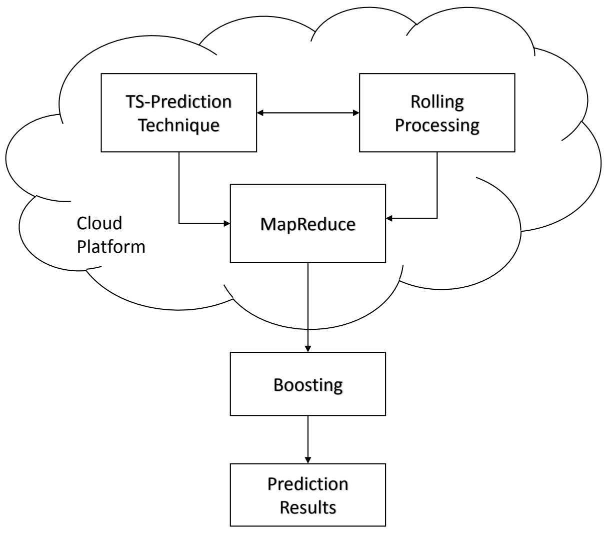


Figure 2.1: Overview of a cloud-based prediction system

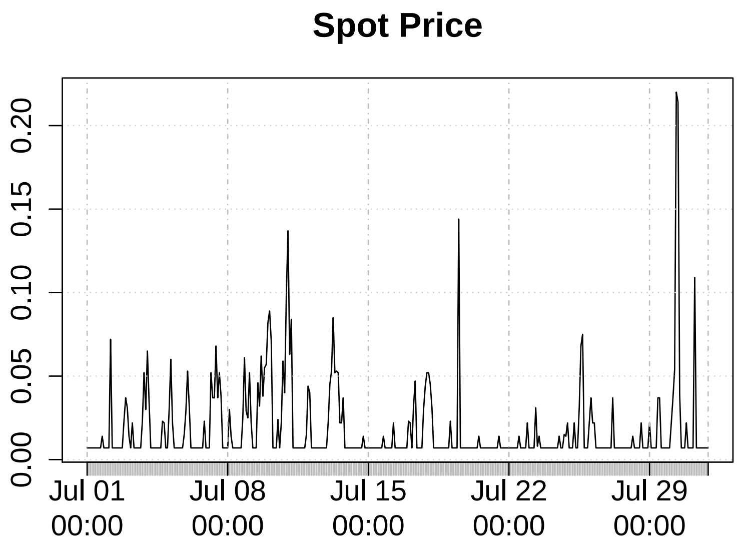


FIGURE 2.2: An example of a spot price time series plot

2.1 Time Series Prediction

2.1.1 Time Series

A time series is defined as a sequence of data points observed typically at successive intervals in time [8]. It can be expressed as an ordered list: $Y = y_1,\ y_2,\ \dots,\ y_n$ [10]. Time series data is extensively used in many disciplines including statistics analysing, signal processing, weather forecasting, biology, mathematical economics and business management [2, 11, 12].

Figure 2.2 illustrates a spot price time series plot. The data are hourly aggregated spot prices of a small Linux AWS EC2 instance in the US east region for the one month period of July 2013. Notice that the data points have been connected through smoothing lines, which make it easier to follow the ups and downs over the time. The spot price for this particular EC2 instance fluctuates randomly.

Many time series can be decomposed into four different components: the long term trend, seasonal components, irregular cycles, and random fluctuations [13]. In Figure 2.3, the time series data are analytically decomposed into components, which are the quarterly retail trade index of 17 European countries from 1996 to 2011. In time series, adjacent observations are in a natural temporal ordering. This intrinsic feature of the time series makes its analysis dependent on the order of the observations, and distinct from other common data, in which there are no dependencies of the observations, such as contextual data [8].

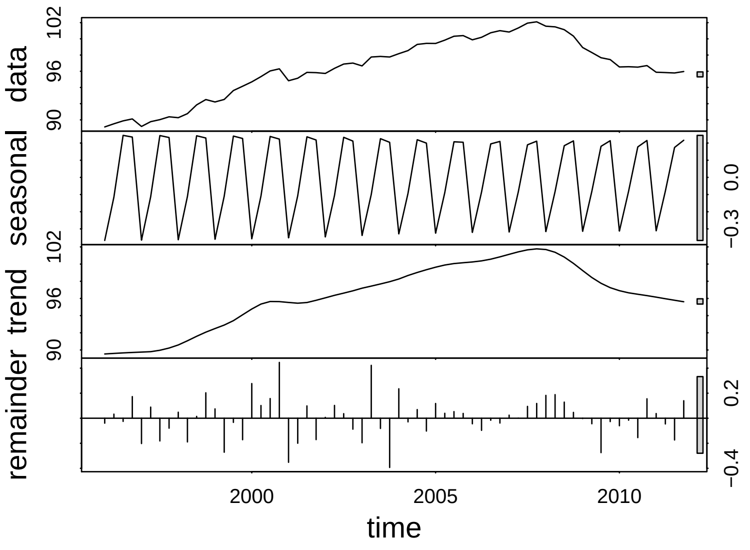


Figure 2.3: Components of time series dataset

Time series analysis is defined as the methods for analysing the characteristics of time series data and extracting meaningful statistical information [14]. Time series forecasting is an important part of time series analysis, in which a model is used to predict future values based on previously observed values [15].

2.1.2 Time Series Prediction

Time series prediction is the use of past and current observations at time t to forecast future values at time $t + l$, where l is the horizon of prediction [8].

Linear time series models are well explored, with auto-regressive (AR) and moving average (MA) models being central to modern stationary time series analysis. Hyndman and Khandakar developed a R library, named *forecast*, for automatic time series forecasting [3]. In this package, some of popular forecasting algorithms are introduced which are principally based on exponential smoothing and autoregressive integrated moving average (ARIMA) models. The automatic forecasting algorithms of the *forecast* select the appropriate time series model, estimate its parameters and then use it to predict the future values. [3]. Furthermore the *forecast* package contains robust algorithms that automatically deal with time series seasonal patterns and random fluctuations.

Figure 2.4 shows an example of a time series prediction using an ARIMA model. The example shows the quarterly beer production in Australia from 1958 Q1 to 2008 Q3, with the prediction objective of the next 20 quarters' production using an ARIMA(1,1,2) model. The details of an ARIMA model are defined in Section 3.4.2.2.

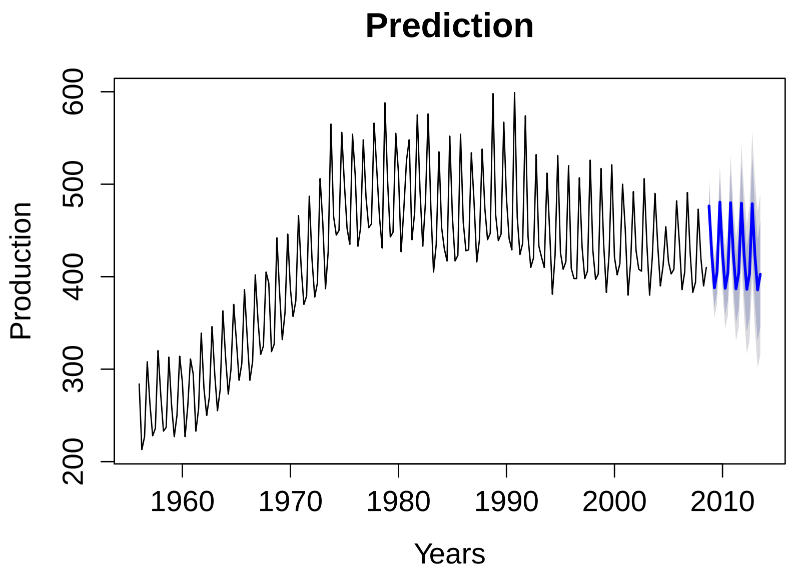


Figure 2.4: Time series prediction using ARIMA model

As the non-linear and non-stationary components often exist in real world time series [16], non-linear approaches such as non-linear autoregressive processes, bilinear models and threshold models are developed and widely used for time series modelling. The generalised autoregressive conditional heteroskedasticity (GARCH) model is a non-linear time series model used to represent the changes of variance over time (heteroskedasticity) [17], which is an extension of autoregressive conditional heteroskedasticity (ARCH) [18]. ARCH and GARCH are used for the volatility of time series data in financial applications, but not studied in this thesis.

Financial time series prediction, however, is a special case as it is statistically different from other time series analysis. Its empirical time series usually contain a high degree of unpredictability, due to the existence of uncontrollable factors and potential or hidden risks influencing the financial markets. For example, the price of a fluctuating stock, which are truly random and not directly predictable, can be modeled as random walks. The theory of random walk states that, in a stock market, using the past observations of a stock price cannot predict its future movement [19, 20]. In the efficient market hypothesis (EMH), it is stated that market efficiency also has some reflections about the uncertainty of the future [21, 22].

In this thesis three pure time series models were used, namely ARIMA, naive and exponential smoothing state space model (ETS), for the purposes of comparison. The drawback of model based approaches is that usually a priori assumption of the underlying distribution of data is required for model parameter estimation. Machine learning techniques can alleviate this issue and cope with the inherent non-linear and non-stationary nature of real world time series.

2.1.3 Rolling Time Series Processing

Different dynamic and statistical methods are available for time series prediction [11]. Commonly, time series prediction algorithms operate on a *rolling window* scheme. Let $\{y_i\}, i = 1, \ldots, N$ be a sampled, discrete time series of length N. For a given integer window size $0 < W \leq N$ and all indices $W \leq k \leq N$, the h-step, $h > 0$, rolling (or sliding) window predictions, $\{\hat{y}_{k+h}\}$ are computed:

$$\hat{y}_{k+h} = f(y_{k-W+1}, \ldots, y_k) \tag{2.1}$$

where f is a prediction algorithm. $\hat{y}_{k+h}$ is approximated such that its error relative to y_{k+h} is minimised.

Rolling analysis of time series is usually applied to dynamically update the parameters of a model. It is a common form to compute the parameter estimates through a window of sample data which is sliding in a fixed length. The estimates of model parameters over the rolling windows should be similar if the data are stationary. On the other hand, if the parameters change frequently or at some observations, the rolling window analysis is able to detect the instable changes over the estimations [23]. As a result, rolling window analysis is often used for the backtesting of the historical time series data, so as to evaluate the stability of forecasting methods and improve the overall prediction accuracy [24]. The first step is to split the initial historical data into two parts, the estimation sample and the other sample for prediction. Then a statistical model is fitted into the estimation sample to forecast a k-steps ahead prediction for the prediction sample [23]. The error measures can be deployed for calculating the difference between k-step ahead prediction and the observed prediction sample of historical data. By repeating the last two steps, the estimation sample is continually processed ahead with certain fixed rolling window length until it reach the end of historical estimation data sample [24]. In the last step, all the predictive results of each single window are then summarised to calculate more statistics, such as the overall k-steps prediction errors, to evaluate the parameters and prediction accuracy of the selected model. The moving average methods are often applied in rolling window analysis to conduct and evaluate the statistical analysis of financial time series [24].

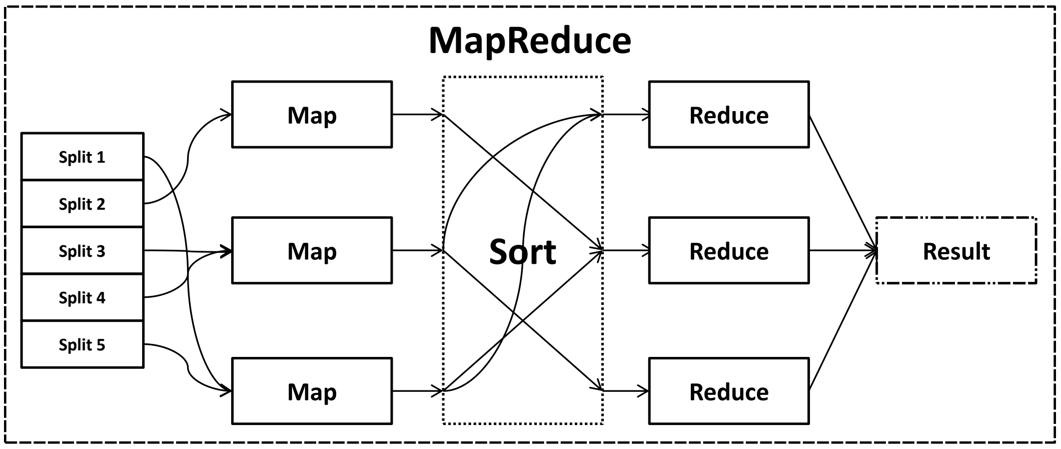


FIGURE 2.5: Overview of parallel processing within a MapReduce environment.

2.2 Parallel Computing

Parallel computing is defined as the simultaneous use of multiple computing resources to solve a computational problem [25]. The precondition of parallelism is that the problem is able to be broken apart into small parts and be processed simultaneously. The execution time with multiple computing processors is always expected less than with a single central processor [25]. Parallel computing has been applied in various areas to improve the computation speed, such as data mining, signal processing and computational simulation ranging from science to financial market.

2.2.1 Parallel in Financial Applications

With the increasing scale of stored transaction data in financial area, there are more and more concerns about parallel computing for financial analysis, in order to optimise business and marketing decisions. Many applications of quantitative finance are able to be parallelised, such as hedging, risk management and portfolio optimization [26]. Therefore, the effective parallel computing modelling and methods are required urgently for financial time series analysis, in order to be competitive in the speed scaling [25]. Currently MapReduce is one of the popular parallel computing mode for large-scale data computation.

2.3 MapReduce

The MapReduce programming model in its current form was proposed by Dean [27]. It centres around two functions, *Map* and *Reduce*, as illustrated in Figure 2.5. The first of these functions, Map, takes an input key-value pair, performs computations and produces a list of key/value pairs as output, which can be expressed as $(k1, v1) \rightarrow list(k2, v2)$. The Reduce function, expressed as $(k2, list(v2)) \rightarrow list(v3)$, takes all of the values associated with a particular key and applies computations to produce the results. Both Map and Reduce functions are designed to run concurrently and without any dependencies. To ensure that each Reduce receives the correct key, an intermediate sort step is introduced. Sort takes the $list(k2, v2)$ and distributes the keys to the appropriate Reducers. The name of MapReduce originated from a technical paper of Google [27] but after that became a standard term to describe this form of parallel processing.

The word count problem is often taken as the classic example to explain how MapReduce solves the real-world problem. The context is associated to the issue of counting the number of occurrences of particular words in a dictionary or big document [27]. The pseudocode example of MapReduce model is listed below:

```
map(String key, String value)
// key: document name
// value: document contents
 for each word w in value
   EmitIntermediate(w, "1")

reduce(String key, Iterator values):
// key: word
// values: a list of counts
  for each v in values:
    result += ParseInt(v);
    Emit(AsString(result));
```

This is the original model form from Dean's paper [27]. The *map* is responsible for adding 1 to the count of occurrences where each word appears; The *reduce* function sums up all the counts for each single word.

Currently the implementation of MapReduce programming model has been applied in various areas to solve the real-word problems, such as the parallel computations of big data across cluster machines. Using MapReduce to count student names in parallel is practical: the Map procedure is responsible for sorting students by first name, and store each student name into queues. In the Reduce procedure, the counting number of students' name and the frequency of each name is summarised. The MapReduce programming model can also be called infrastructure or framework. It is an abstraction to the complex parallel computing procedures and data distributions [28]. By using

MapReduce, all the data transfers and interaction between each single part of the system is becoming manageable, and also data redundancy and fault tolerance are considered.

Dean designed the concept of MapReduce from the basic map and reduce functions in functional programming [27]. However, it has evolved and extended from their original forms and now has more powerful functionalities. The Map and Reduce functions are not the only major contribution, scalability and fault-tolerance are key to solving large-scale problems on commodity computing equipment [27]. The implementation of MapReduce in a single node or thread is not going to be faster than a traditional sequential computing. Only deploying MapReduce on a large cluster becomes beneficial for the optimisation of distributed operation, cost reduction of network communication and fault tolerance.

Currently, MapReduce concepts have been applied in many areas and its libraries have been ported to various programming languages. The Apache Hadoop is one of the most popular implementations.

2.3.1 Hadoop

Apache Hadoop is an implementation of the MapReduce framework for cluster computers constructed from computing nodes connected by a network. It was created by Doug Cutting and Mike Cafarella in 2005 [29], and operates under the Java Runtime Environment (JRE) which ensures portability across platforms [30].

Used by 60% of the Fortune 500 companies, Hadoop has become the industry standard for dealing with big data problems. The Hadoop implementation of MapReduce can be described as a cluster of TaskTracker nodes, dealing with a JobTracker and client node, see Figure 2.6. Once a MapReduce application has been created, the job is committed to Hadoop and then passed to the JobTracker which initialises it on the cluster. During execution, the JobTracker is responsible for managing the TaskTrackers on each node and each TaskTracker spawns Map and Reduce tasks depending on the JobTraker's requirements [27]. Inputs to the Map tasks are retrieved from the Hadoop distributed file system (HDFS), a shared file system that ships with Hadoop. These inputs are partitioned into multiple splits which are passed to the map tasks. Each split contains a small part of the data that the Map function will operate on. The Map results are sorted and passed to the Reduce tasks. The results of the Reduce tasks are written back to HDFS where they can be retrieved by the user [6].

In a small Hadoop cluster, there is only one single master with multiple slave nodes. As shown in Figure 2.6, the master contains JobTracker, TaskTracker, NameNode and

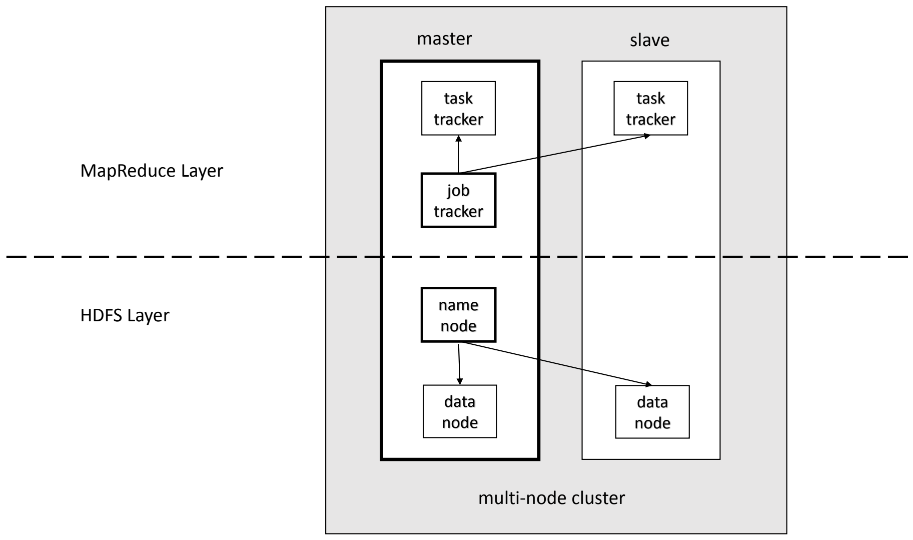


FIGURE 2.6: Overview of a multi-node Hadoop cluster.

DataNode. But the slave nodes only contains a DataNode and TaskTracker. They are managed and controlled by the NameNode and JobTracker of master node. The small-scale Hadoop cluster is used only in nonstandard applications [30].

For a larger size cluster, a dedicated NameNode server is assigned to manage HDFS with file system index. The secondary NameNode duplicates the structure of master NameNode as a snapshot. This structure can prevent the file system corruption and reduce the risk of data loss [30]. Similarly the JobTracker server is responsible for job scheduling.

2.3.2 Hadoop Distribution File System

The Hadoop distributed file system (HDFS) is a shared file system developed for the Hadoop framework. In a Hadoop cluster, the NameNode and DataNodes are formed in the HDFS layer (see Figure 2.6). Usually the master has both of nodes and slave node only has DataNode, because of the DataNode is not required to be present in each node. The TCP/IP layer is used by the file system for communication between nodes. The interaction of each DataNode is accomplished by using the protocol specific to HDFS [31].

HDFS is distributed, scalable, and portable. It is usually used to stores big data files over multiple machines, which typically can be in the range of gigabytes to terabytes [32]. HDFS achieves reliability by replicating the data files across different nodes. By default, 3 replications of data files are stored on 3 multiple nodes: two copies are on the same rack, and another one is on a different rack [33]. The data nodes are interactive and can reform the data rebalancing.

Using HDFS provides a significant data awareness in file system. The responsibility of JobTracker is to assign the Map or Reduce jobs to TaskTrackers. The data location is aware of while scheduling the jobs. More specifically, each node of the cluster only schedules the Map or Reduce tasks on its own data [33]. For example, node M contains data (a, b) and then node M would only be scheduled to perform Map or Reduce tasks on (a, b). This advantage prevents the unnecessary traffic transfer over the cluster nodes, and reduces the data traffic time [33]. However, this advantage is now always available when Hadoop is used with other file systems. Moreover, Jiong Xie et al. [34] discovered that it significantly impacts the job completion time, demonstrated by running intensive-scale jobs.

HDFS was initially designed for most files except the systems requiring concurrent write-operations [35]. In addition a Filesystem in Userspace (FUSE) interface is included into HDFS, enabling users to write a normal userland application as a bridge for a traditional filesystem interface [36].

2.3.3 HBase

HDFS file system is also the basis of Apache HBase, a column-oriented distributed database management [37]. HBase has become the standard tool for big data storage and query. It originates from Google's BigTable and is developed as part of the Apache Hadoop project [6]. The instinctive features of HBase are providing the capabilities of querying and storing big data for Hadoop, such as serving database tables as the input and output for MapReduce jobs and real-time data access. Additionally HBase features file compression, in-memory operation and bloom filters [38].

2.3.4 Rhipe Package

Rhipe is a R package that integrates Hadoop within the R programming environment [39]. In other words, Rhipe is a fusion of R and Hadoop, combining the interactive R environment and the highly scalable parallel Hadoop framework, to facilitate the statistical analysis of complex big data [40]. Rhipe uses Hadoop to parallelise the computationally intensive tasks.

This package was developed by Saptarshi Guha from the Purdue Statistics Department [40]. Currently a core development team is established and a Google discussion group is provided to all the users and researchers [39].

The impressive contribution of Rhipe is that its functionalities are achievable with small-scale data sets [40]. It was initially inspired by two goals [41]. The first goal is to achieve deep data analysis in an efficient way. Moreover it is more concerned about avoiding data loss which caused by improper data reductions. In order to achieve the first goal for small or big data, the visualised and statistical methods are required to extract the characteristics of data and detailed statistics. Therefore, the second goal is to integrate with the high-level R language, in order to improve the efficiency and effectiveness by avoiding low level programming.

The Rhipe chooses Hadoop to access scalable I/O and parallel the computing. As described above in section 2.3.1, Hadoop was designed for cluster machines to handle the comprehensive computing in a scalable way. It is practical to deal with very different performance characteristics of different operating platforms [30]. The advantages of integrating Hadoop is not limited to the parallelism of cluster computing over time, but also the detailed tracking record in its open-source application which supported by Apache [42].

Compared to the existing parallel R packages, Rhipe is more beneficial for users in data analysis [40]. In addition, Rhipe is more computationally effective by applying the high capabilities and support of Hadoop. More details are presented in Chapter 4 section 4.3.1.

2.4 Amazon Web Service

Amazon Web Services (AWS) is a cloud computing platform with a collection of remote computing services, served over the Internet. The most well-known services of AWS are Amazon Elastic Compute Cloud (EC2) and Amazon Simple Storage Service (S3) [43]. The significant advantage of utilising AWS is providing an elastic computing service with high capacity. Using AWS services requires less resources and is cheaper than establishing a cloud server in-house.

2.4.1 Elastic Cloud Computing

Amazon Elastic Compute Cloud (EC2) is a pivotal part of AWS cloud platform [44]. The AWS users can rent virtual EC2 computers for computing tasks, building applications and servers. EC2 also provides the scalable deployment by using AWS' Amazon Machine Image (AMI) service, which allows user to create and manage a virtual machine with user desired software. Each virtual EC2 machine created by users is called an "instance" [44].

The “elastic” feature of EC2 can be explained as follows: users can create, launch, and terminate the cloud instances based on their needs and only pay for the cost of active running hours of instances. There are three basic EC2 instance types, namely On-Demand, Reserved and Spot [44]. All types of the instances provides the same standard computing capacity although they may be from different data centres based on the different geographical locations. The only difference between these three types is the different pricing schemes of the instances [44].

In November 2010, Amazon switched its own retail website to EC2 and AWS [43].

2.4.2 StarCluster

StarCluster is a cluster computing toolkit, specially designed for Amazon’s Elastic Compute Cloud (EC2). It is open-source and released under the LGPL license [45].

Using StarCluster, users are able to build, configure and manage the AWS virtual machine clusters in a more automatic way. Additionally, StarCluster allows users to create a cloud computing environment for parallel computing quickly and easily trough simple operations. The target users group of the StarCluster is the academic researchers with the needs for cluster computing services [45].

StarCluster is a command-line tool written in Python with an user-friendly interface to AWS EC2 [45]. The most beneficial part of StarCluster is that the strong supports of variants EC2 Linux system images.

There are three reasons to build StarCluster specially for AWS [45]. Firstly cloud computing is the future trend for computing service, allowing all the intensive programming works outsourced. AWS is playing the lead role among the existing popular and standard cloud computing platforms. Secondly, in contrast to the different controlling and configuration manager for different AWS services, there is a need for a simple control method over all different commands through a programmable API [45]. Furthermore, with the helps of StarCluster toolkit, systems administrators and programmers can focus more on the researches with a comfortable cloud user environment, rather than spending time on the complicated procedure of managing a big cluster. All of these logistical complications are removed with the development of this easy-to-use command toolkit with an easier access to AWS cloud computing service.

More practical benefits of StarCluster will be introduced and demonstrated in Section 4.2.4.

2.5 Boosting

Boosting is a general method to improve the accuracy of a given set of learning algorithms [46]. The idea of boosting is to combine a set of learners to form an ensemble in order to achieve a better performance. Assuming that the learning hypotheses can be presented as $h_1, h_2, \ldots, h_T$, and the ensemble hypothesis is a sum of these hypotheses [47]:

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x). \tag{2.2}$$

The parameter α_t is the coefficient of each combined ensemble member h_t. The learner h_t is learned with the boosting procedure through the interoperation of α_t. Therefore, the hypothesis boosting problem can be simplified and referred to the process of combing a set of weak hypothesises into a strong hypothesis [48].

Boosting was inspired by a machine learning theory called the “PAC” (Probably Approximately Correct) learning model [49], due to Valiant’s the Learnable Theory [50]. Professor Michael Kearns was the first to pose the question “Can a set of weak learners create a single strong learner?” in his hypothesis [48]. Later boosting theory proved that if each base learner performs slightly better than random guess, it is possible to combine them to generate an arbitrarily better performance.

Schapire was the first to provide a polynomial time boosting algorithm [51]. Later he applied the boosting idea to a real-world problem, using the base learners of neural networks for boosting [52].

After the above works appeared, boosting was defined as a learning algorithm, which can generate high-accuracy predictions or estimates using a set of base learners, which in turn can efficiently generate hypotheses slightly better than random guess [53]. In machine learning, a weak learner is defined as a classifier only slightly correlated with the actual target. In contrast, a strong learner is a classifier that is well-correlated with the actual target [50].

A boosting algorithm can be applied to model fitting, variable selection, and model choice [54]. Compared to the original outcomes from variant learners, the outcome of boosting always leads to a better prediction or estimation. In order to improve the predictive quality, boosting is usually considered as an efficient but time-consuming approach for increasing the accuracy of forecasting. More practical theories are introduced and analysed in Section 5.1.

2.6 Related Work

Both the processing of times series data and specific time series prediction techniques have been previously studied by different researchers. Hadoop.TS [55] was proposed in 2013 as a toolbox for time series processing in Hadoop. This toolbox introduced a bucket concept which traces the consistency of a time series for arbitrary applications. Sheng et al. [56] implemented the extended Kalman filter for time series prediction using the MapReduce methodology. The framework calculated the filter's weights by performing the update step in the Map functions, whilst the Reduce function aggregated the results to find an averaged value.

A Hadoop based ARIMA prediction algorithm was proposed and utilised for weather data mining by Li et al. [57]. The work describes a nine step algorithm that employs the Hadoop libraries, HBase and Hive, to implement efficient data storage, management and query systems. Stokely et al. [58] described a framework for developing and deploying statistical methods across the Google parallel infrastructure. By generating a parallel technique for handing iterative forecasts, the authors achieved a good speed-up.

The work presented in this thesis differs from previous work in three key areas. First, it presents a low-overhead dynamic model for processing irregularly sampled time series data in addition to regularly sampled data. Secondly, the proposed framework allows applying multiple prediction methods concurrently as well as measuring their relative performance. Finally, it offers the flexibility to add additional standard or user-defined prediction methods that automatically utilise all of the functionalities offered by the framework.

In summary, the proposed framework is mainly focused on effectively applying the Hadoop framework for time series rather than the storage of massive data. The main objective of this thesis is to present a fast prototyping architecture to benchmark and backtest rolling time series prediction algorithms. To the best of authors knowledge, this is the first study focusing on a systematic framework for rolling window time series processing using MapReduce methodology.

Chapter 3

Rolling window time series prediction using MapReduce

To achieve the parallelism of rolling processing in time series, this chapter proposes a methodology to facilitate retrieval and rolling window prediction of irregularly sampled large-scale time series data, using MapReduce Framework. Although using Hadoop in the traditional way is not suitable for rolling analysis, there are still variant advantages of implementing the rolling time series prediction in Hadoop, which is considered as the best option so far. Time stamps is the unique feature of time series data, which can be used as an indicator/indexer of locating the indices of data. The notion of index pool is designed to locate the overlapping data which across two adjacent windows. Special issues of rolling analysis in time series data are further discussed and a straightforward implementation is proposed to deal with all issues, resulting in a significant improvement on efficiency.

3.1 Issue of Rolling Analysis using Hadoop

As stated in Chapter 2, the rolling time series processing is to fit the target algorithm on the sample data of a fixed window length. The unique feature of time series data is that the analysis depends on the order of time series observations. Specifically, the sample data in each window is partially overlapping with the adjacent windows. This intrinsic feature does not influence the predictions of rolling window time series sequentially, due to that the entire sample data is accessible to be partitioned into splits. MapReduce is originally designed to run Map and Reduce functions concurrently and without any dependencies. Therefore, a technique is needed to locate the overlapping data from neighbouring data file precisely and assemble them. This is the core difficulty associated with this research. The time series bucket concept [56] was the major inspiration in solving this issue with the notion of an *index pool*.

3.2 Proposed Methodology

While financial time series are often recorded in irregular ticks, many forecasting algorithms require a periodic time series as input. In order to make a time series periodic and/or reduce its temporal resolution, an optional normalization of data may be required in preprocessing stage. This is achieved by applying a user supplied algorithm to rolling windows of the aggregated data.

In the proposed framework, it is assumed that the numerical samples and their timestamps are stored on HDFS before invoking the input data. Then *index pool* is introduced, which is a table of time series indexing and time-stamp information for the entire file directory stored on the HDFS. The major contribution of *index pool* to the entire framework is assigning the appropriate index keys to time series entries, to guarantee an appropriate and precise distribution of data across multiple computational node, while handling time series dependencies. As a result, the index pool is considered the core of the architecture.

Data aggregation and pre-processing are handled by the *map* function. The aggregated data is then indexed using the index pool and is assigned a key, such that window parts spread across multiple splits are assigned the same unique key. If all of data for a window is available in the *map* function, the prediction is performed and its performance (in terms of prediction error) is measured; the prediction result along with its keys is then passed to the *reduce*. Otherwise, the incomplete window and its key are directly passed to the *reduce*, where they are combined accordingly with the partial window from other splits into proper rolling windows. Prediction for these windows is performed in the *reduce*. The final output is created by combining the results of all prediction.

The pseudocode below illustrates the detailed steps of proposed algorithm in both of *map* and emphreduce stages, which process the rolling window prediction with window length l.

Map Stage:

```
for data file in $[file_1, \dots, file_m]$ do
    $f_{ID}$ = file ID in index pool
    fetch the index list corresponding $f_{ID}$
    partition the sample data into windows with $l$ length
    for window $W$ in $[W_1, \ldots, W_m]$ do
        if $W$ is a complete window then:
            predict $f(W, otherparams)$
            return $< K, Predictions >$ to Reduce
        else
            return $< K, IncompleteWindows >$ to Reduce
        end if
    end for
end for
```

Reduce Stage:

```
for $< K, V >$ in Map Results do
    if for the same $K$ has multiple vales $V$ then:
        assemble the incomplete windows together and sort in right order
        predict $f(W, otherparams)$ with the assembled complete window
        return $< K, Predictions >$
    else
        reorder the results with key $K$
    end if
end for
```

Figure 3.1 presents the work flow and the architecture of the proposed system. There are five stages in the entire system working flow. The Data storage procedure is processed in HDFS; Mapper and Reducer functions are responsible for the rolling processing of time series prediction; the final outcomes and error measures are taken in Finalisation stage; finally there is a post-processing step to facilitate the boosting of predictions.

Figure 3.2 shows how the data flows through the blocks. As the logistic design in both Map and Reduce stages, the different procedures of rolling processing facing complete and incomplete windows are clearly illustrated in the diagram. The complete windows are being processed straightforward in Map stage; however, incomplete windows are assembled and process afterwards in Reduce stage. All results are sorted and aggregated in Finalisation session of the working flow.

The rest of this section describes the system components in greater detail, and two different architecture designs are discussed later. The last step of the architecture, boosting, is described separately in Chapter 5.

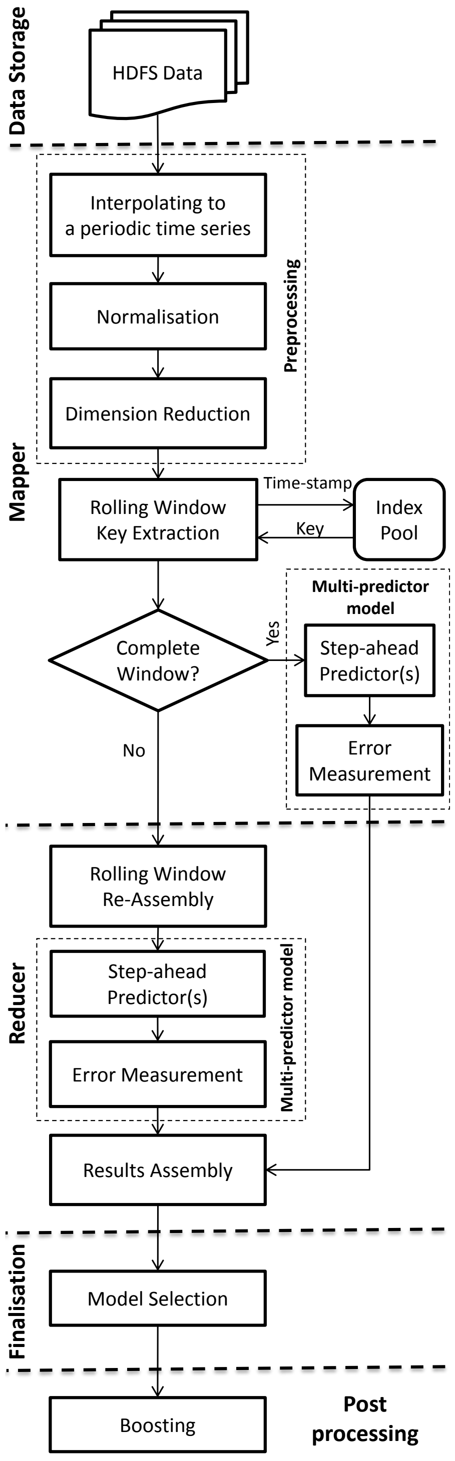


Figure 3.1: The system's architecture

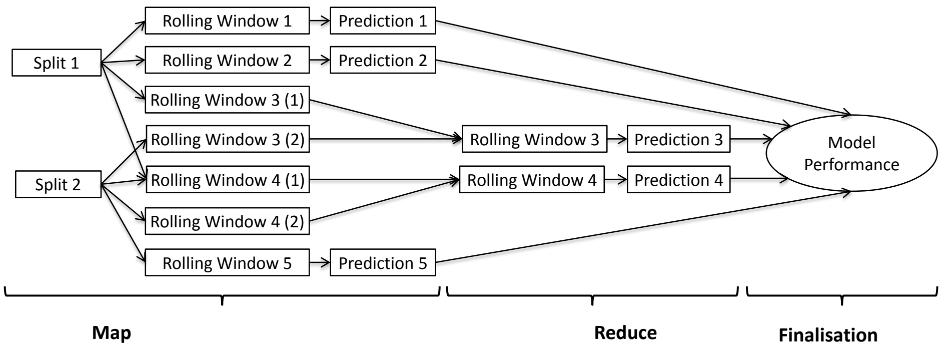


FIGURE 3.2: Flow of data in the proposed framework.

3.3 Design

3.3.1 Data Storage and Index Pool

In the proposed system, the assumption that time series are stored sequentially in multiple files is made. Stored files cannot have overlapping time-stamps and are not necessarily separated at regular intervals, and the lengths of the files are not necessarily equal. Each sample in time series contains the data and its associated time stamp. The name of each file stored in HDFS is the first time-stamp in a ISO 8601 format, which can easily locate the target files and data access.

The major properties of Index pool designed in the proposed system are only File Name, Start Time-stamp, End Time-stamp and Index List (the length of the files). These basic components can be retrieved, and other additional components could be added if required for special needs. It is represented as a global table that can be accessible over the entire processing procedure, as stored in same file directory (HDFS for example). This Index pool significantly improved the traceability of time series sample under rolling analysis, in order to avoid the loss of overlapping data across multiple files. Table 3.1 shows an example of an *index pool*.

TABLE 3.1: Example of an index pool

File Name	Start Time-stamp	End Time-stamp	Index List
2011-01-01	2011-01-01 00:00	2011-01-01 22:00	$1 \rightarrow 12$
2011-01-02	2011-01-02 00:00	2011-01-02 22:00	$13 \rightarrow 24$
2011-01-03	2011-01-03 00:00	2011-01-03 22:00	$25 \rightarrow 36$
2011-01-04	2011-01-04 00:00	2011-01-04 22:00	$37 \rightarrow 48$
2011-01-05	2011-01-05 00:00	2011-01-05 22:00	$49 \rightarrow 60$

The index pool enables arbitrary indices to be efficiently located and is used to detect and assemble adjacent windows. Interaction of the index pool with the MapReduce framework is illustrated above in Figure 3.1

Index pool creation is performed in a separate maintenance step prior to forecasting. Assuming that data can only be appended to the filesystem (as is the case for HDFS), index pool updates are trivial, as time series data is a continuous signal in real world.

3.3.2 Preprocessing

Work in the *map* function starts by receiving a split of data. A preprocessing step is performed on the data, with the following goals:

- Creating a periodic time series: In time series prediction, it is usually expected that the sampling is performed periodically, with a constant time-difference of Δt between consecutive samples. If the input data is unevenly sampled, it is first interpolated into an evenly sampled time series. Different interpolation techniques are available, each with their own advantage [59]. Linear interpolation is one of the commonly used techniques [60].
- Normalisation: Many algorithms require their inputs to follow a certain distribution for optimal performance. Normalisation preprocessing adjusts statistics of the data (e.g. the mean and variance) by mapping each sample through a normalising function.
- Reducing time-resolution: Many sampled datasets include very high frequency data (e.g. high frequency trading), while the prediction use-case requires a much lower frequency. Also *the curse of dimensionality* prohibits using high dimensional data in many algorithms. As a result, users often aggregate high frequency data to a lower dimension. Different aggregating techniques include averaging and extracting open/high/low/close values as used in financial Technical Analysis from the aggregated time frame.

3.3.3 Rolling Windows

Following preprocessing, the *map* function tries to create windows of length W from data $\{y_i\}, i = 1, \cdots, l$, where l is the length of data split. As explained earlier, the data for a window is spread across 2 or more splits starting from the sample $l-W+1$ onwards and the data from another *mapper* is required to complete the window.

To address this problem, the *map* function uses the index pool to create *window index keys* for each window. This key is globally unique for each window range. The *map* function associates this key with the complete or partial windows as tuple $(\{y_j\}, k)$, where $\{y_j\}$ is the (partial) window data and k is the key.

In the *reduce*, partial windows are matched through their window keys and combined to form a complete window. The keys for already complete windows are ignored. In Figure 3.2, an example of partial windows being merged is shown.

In some cases, including model selection and cross-validation, there is no need to test prediction algorithms on all available data; correspondingly the *map* function allows for arbitrary strides in which every mth window is processed.

3.3.4 Prediction

Prediction is performed within a *multi-predictor model*, which applies user all of supplied predictors to the rolling window. Each data window $\{y_i\}, i = 1, \cdots, w$ is divided into two parts: the training data with $\{y_i\}, i = 1, \cdots, w - h$, and $\{y_i\}, i = w - h, \cdots, w$ as the target. Separation of training and target data at this step removes the possibility of *peeking* into future from the architecture.

The training data is passed to user supplied algorithms and the prediction results are returned. For each sample, the time-stamp, observed value and prediction results from each algorithm are stored. For each result, user-defined error measures such as an L1 (Manhattan) norm, L2 (Euclidean) norm or relative error are computed.

To reduce software complexity, the initial design is to perform all the predictions in the *reduce*, regardless of the concerns whether the sample data is from an incomplete window or complete window; however, this *straightforward* method is inefficient due to the MapReduce architecture. In the Result section there is a detailed comparison of these two designs and a demonstration on the advantages of proposed design. Therefore in our proposed framework only partial windows are predicted in the *reduce* after reassembly. Prediction and performance measurement of complete windows are performed in the *map*, and the results and their index key are then passed to the *reduce*.

3.3.5 Finalisation

In the *reduce*, prediction results are sorted based on their index keys and concatenated to form the final prediction results. The errors of each sample are accumulated and converted to a summary error measures, allowing model comparison and selection.

Commonly used measures are Root Mean Square Error (RMSE) and Mean Absolute Prediction Error (MAPE):

$$\text{RMSE}(Y, \hat{Y}) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{3.1}$$

$$\text{MAPE}(Y, \hat{Y}) = \frac{1}{N}\sum_{i=1}^{N}|\frac{y_i - \hat{y}_i}{y_i}| \tag{3.2}$$

where $Y = [y_1, \cdots, y_N]$ is the observed time series, $\hat{Y} = [\hat{y}_1, \cdots, \hat{y}_N]$ is the prediction results and N is the length of the time series.

Akaike Information Criterion (AIC) is another measure, and is widely used for model selection. AIC is defined as:

$$\text{AIC} = 2k - 2\ln(L) \tag{3.3}$$

where k is the number of parameters in the model and L is the likelihood function.

3.4 Forecasting

3.4.1 Multi-Predictor Model

In this section, some of popular time series prediction algorithms are described, which are used as the predictor models of the experimental results in Chapter 6. In addition, a multi-predictor model (MPM) scheme is applied to the proposed framework for the automatic selection of an appropriate prediction model. MPM is expected to improve the efficiency of batching predictors. In this scheme, the multi-predictor function is called in the Map or the Reduce, which in turn applies all user supplied predictors to each data window, returning a vector of prediction results (and error measures) for every predictor. Following the MapReduce, MPM selects the most proper predictor for the particular test time series by comparing the error measures (RMSE and MAPE) of each prediction model, which are illustrated in Finalisation section.

3.4.2 Linear Autoregressive Models

Autoregressive (AR) models are a type of statistical process where any new sample in a time series is a linear function of its past values. Because of their simplicity and generalisability, AR models have been studied extensively in statistics and signal processing and many of their properties are available as closed form solutions [11].

3.4.2.1 AR model

A simple AR model is defined by:

$$X_t = c + \sum_{i=1}^{p} \phi_i X_{t-i} + \epsilon_t \tag{3.4}$$

where X_t is the time series sample at time t, p is the model order, $\phi_1, \ldots, \phi_p$ are its parameters, c is a constant and ϵ_t is white noise.

The model can be rewritten using the backshift operator B, where $Bx_t = x_{t-1}$:

$$(1 - \sum_{i=1}^{p} \phi_i B^i) X_t = c + \epsilon_t \tag{3.5}$$

Fitting model parameters ϕ_i to data is possible using the least-squares method. However, finding the parameters of model for data $X_1, \ldots, X_N$, requires the model order p to be known in advance. This is usually selected using AIC. First, models with $p \in [1, \ldots, p_{max}]$ are fitted to data and then the the model with the minimum AIC is selected.

To forecast a time series $X_1, \ldots, X_N$, first a model is fitted to the data. Using the model, predicting the value of the next time-step is possible by using Eq. 3.4.

3.4.2.2 ARIMA models

The autoregressive integrated moving average (ARIMA) model is an extension of AR model with moving average and integration. An ARIMA model of order (p, d, q) is defined by:

$$\left(1 - \sum_{i=1}^{p} \phi_i B^i\right) (1 - B)^d X_t = c + \left(1 + \sum_{i=1}^{q} \theta_i B^i\right) \epsilon_t \tag{3.6}$$

where p is autoregressive order, d is the integration order, q is the moving average order and θ_i is the ith moving average parameter. Parameter optimisation is performed using Box-Jenkins methods [11], and AIC is used for order selection.

AR(p) models are represented by ARIMA$(p, 0, 0)$. Random walks, used as Naïve benchmarks in many financial applications are best modelled by ARIMA$(0, 1, 0)$ [61].

3.4.3 NARX Forecasting

Non-linear auto-regressive models (NARX) extend the AR model by allowing non-linear models and external variables being employed. Support vector machines (SVM) and Artificial neural networks (ANN) are two related class of linear and non-linear models that are widely used in machine learning and time series prediction. More details about the algorithms are going to be elaborated as follows.

3.4.3.1 ETS

ExponenTial Smoothing state space (ETS) is a simple non-linear auto-regressive model. ETS estimates the state of a time series using the following formula:

$$\begin{aligned} s_0 &= X_0 \\ s_t &= \alpha X_{t-1} + (1-\alpha)s_{t-1} \end{aligned} \tag{3.7}$$

where s_t is the estimated state of time series X_t at time t and $0 < \alpha < 1$ is the smoothing factor.

Due to their simplicity, ETS models are studied along with linear AR models and their properties are well-known [11].

3.4.3.2 SVM

SVMs and their extension support vector regression (SVR) use a kernel to map input samples to a high dimensional space, where they are linearly separable. By applying a soft margin, outlier data is handled with a penalty constant C, forming a convex problem which is solved efficiently [62]. As a result, there are several models using SVM that have been successfully studied and used in time series prediction [63].

In this thesis, a Gaussian radial basis kernel function was used:

$$k(x_i, x_j) = \exp\left(\frac{-1}{\sigma^2}||x_i - x_j||^2\right) \tag{3.8}$$

where x_i and x_j are the ith and jth input vectors to the SVM, and σ is the kernel parameter width.

The time series NARX model using SVM is defined as:

$$X_t = f(C, \sigma, X_{t-1}, \cdots, X_{t-w}) \tag{3.9}$$

where f is learnt through the SVM and w is the learning window length.

To successfully use SVM for forecasting, its hyper-parameters including penalty constant C, kernel parameter σ and learning window length w have to be tuned using cross validation. Ordinary cross validation cannot be used in time series prediction as it reveals the future of the time series to the learner [11]. To avoid peeking, the only choice is to divide the dataset into two past and future sets, then train on past set and validate on future set.

The following algorithm is used to perform cross-validation:

```
for w in $[w_{min}, \ldots, w_{max}]$ do
    prepare $X$ matrix with lag $w$
    training_set ← first 80% of $X$ matrix
    testing_set ← last 20% of $X$ matrix
    for $C$ in $[C_{min}, \ldots, C_{max}]$ do
        for $\sigma$ in $[\sigma_{min}, \ldots, \sigma_{max}]$ do
            $f$ ← SVM($C$, $\sigma$, training_set)
            err ← predict($f$, testing_set)
            if err is best so far then:
                best_params ← $(C, \sigma, w)$
            end if
        end for
    end for
end for
return best_params
```

3.4.3.3 Artificial Neural Networks

Artificial neural networks (ANN) are inspired by biological systems. An ANN is formed from input, hidden and output node layers which are interconnected with different weights. Each node is called a neuron.

Similar to SVM, ANNs have been extensively used in time series prediction [64]. In an NN autoregressive (NNAR) model, inputs of the network is a matrix of lagged time series, and the target output is the time series as a vector. Back-propagation is a learning algorithm used to minimise error of this network's output. Basically back-propagation weights the connected neural neuron by gradient decent method. The structure of a back-propagation ANN is shown in Figure 3.3. The solid lines are the forward moves and dot lines are the training connection moves. In the output layer, each neuron's

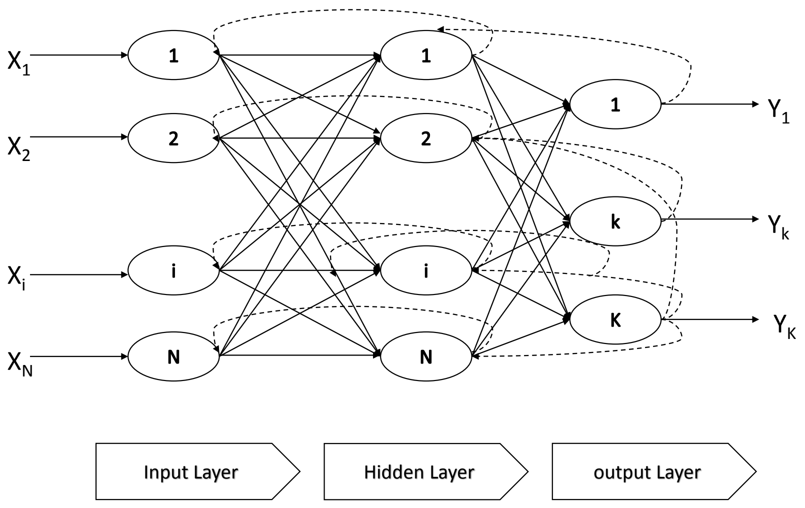


FIGURE 3.3: The back-propagation neural network structure

output is aggregated by the previous level's neurons multiplied by their corresponding weights [65].

Xiao and Chandrasekar successfully applied back-propagation ANNs to solve the e-commerce problems in customers patterning, and estimate the rainfall based on radar data [66]. In this thesis, a feed-forward ANN with a single hidden layer is used for the experiments, described by

$$f(x) = g(\sum_{i=1}^{N} w_i x_i) \tag{3.10}$$

where x_i is the value of neuron and the weights of each level w_i, and $g(y)$ is the activation function. The structure of this ANN is similar to the figure shown in Figure 3.3.

3.5 Summary

In this chapter, the proposed methodology was explained in detail after raising the issue of rolling time series analysis using Hadoop. It includes how the new data storage index design, *index pool*, was formed, and how rolling windows smoothly proceed using MapReduce model. The rest of this chapter presented the multi-predictor model developed for automatic algorithm selection and the details of each forecasting algorithm used in the proposed framework.

Chapter 4

Cloud-based Platform

Another objective of this work is to deploy the the framework on a cloud-based environment. This chapter provides a comparison of the state-of-art in commercial cloud services, including Amazon Web Service and Microsoft (Windows) Azure, particularly with respect to their ability of supporting the large-scale experiments. Other essential aspects such as price and storage services are also considered. The Rhipe software framework is chosen to provide the development environment for the parallelism of the proposed framework. In this chapter, a practical example of Rhipe and its advantages are demonstrated in detail.

4.1 Cloud Platform

The shift to cloud computing is a major change for businesses and industries. Cloud computing is defined as a system where software applications may be run in an environment consisting of a logically abstract network of general purpose computers [67]. Following the definition, there are numerous benefits offered by a cloud-based platform, i.e. (1) lets developers create apps which are available to users anytime and anywhere; (2) provides self-service access to a variety of computing resources; (3) enables an elastic control of resources allocation; (4) only charges for the resources users used.

There are two major use-cases of a cloud-based platform: computing and storage (see Table 4.2). From the development viewpoint, developers currently categorise the different levels of cloud computing services to IaaS (Infrastructure as a Service) and PaaS (Platform as a Service).

4.1.1 Comparison of Different Cloud Platform

Generally a cloud-based platform can be easily utilised without physical room space, regular maintenance, front-end investment in machines and other facilities. As a result of market demand and technological innovations, several cloud platform service providers have been established in recent years.

In this chapter, two of the most prevalent cloud service providers are studied:

- **Amazon Web Services**
  AWS plays a leading role in the constant innovation and enhancements to the service. AWS now provides 30 services ranging from basic cloud computing to real-time data storage all across the world since 2006 [43]. Major services cover the areas of computing, networking, storage & content delivery, database, App services, mobile services and applications. There are a variety of computing resources and instance types offered to meet the unique needs of different user groups. AWS has established 8 data centres all over the world in 2014, in order to provide the standard services to the users in different regions [43].

- **Microsoft**
  Microsoft Azure (formerly called Windows Azure) is a cloud computing platform and infrastructure, owned by Microsoft. It provides the cloud resources, applications and services through a global datacentre network. Both PaaS and IaaS services are supported with many different programming languages, tools and frameworks. Azure was launched in 2010 [68].

Generally these two well-known cloud services provide users with similar and compatible services. They both have user-friendly interface, allowing users to create and manage the cloud computing environment in a few minutes through simple operations on web browsers. There are minor performance differences between these two providers, but often vary in terms of the machine configuration options and pricing schemes.

Table 4.1 and Table 4.2 compare the price and service features between AWS and Microsoft Azure [43, 69]. AWS is slightly cheaper than Microsoft in tiers with similar configuration. All of the compared prices are studied in the same region (US east) and of the same operating system (Linux). Both options provide similar IaaS and PaaS services, and relational, scale-out and blobs storage. AWS is found more preferable due to its lower operational cost.

Table 4.1: Price comparison of Amazon and Microsoft

Instance	**Virtual CPUS**	**RAM**	**Cost per hour**
Amazon m3.medium	1	3.75GB	7 cents
Amazon m3.large	2	7.5GB	14 cents
Amazon m3.xlarge	4	15.00GB	28 cents
Amazon m3.2xlarge	8	30.00GB	56 cents
Microsoft Azure Medium A2	2	3.50GB	15.4 cents
Microsoft Azure Large A3	4	7.00GB	30.8 cents
Microsoft Azure Extra Large A4	8	14.00GB	61.6 cents

Table 4.2: Service comparison of Amazon and Microsoft

Name	Computing		Storage		
	IaaS	PaaS	Relational	Scale-out	Blobs
Microsoft	Hyper-V Cloud	Windows Azure	SQL Azure	Azure Tables	Azure Blobs
Amazon	EC2	Elastic Beanstalk	RDS	SimpleDB	S3

4.2 AWS

4.2.1 Elastic Compute Cloud

Amazon Elastic Compute Cloud (Amazon EC2) was briefly introduced in Chapter 2 which is the central service of AWS, providing resizable computational cloud resources.

AWS EC2 has a simple web service interface for users to manage and control the computing properties easily and quickly [44]. It provides complete controls of the computing resources and lets users run on Amazon's proven computing environment. The time of creating and booting new cloud computing resources is significantly reduced by using AWS EC2. AWS EC2 enables the quick scaling on computing capacity as the user requirements change, hence it is "elastic" [44]. Amazon's EC2 also reduces the monetary cost of computing utilisation by allowing users to only pay for actual running hours they use. Functionally, it provides a variant selection of instances to fit different user requirements. The EC2 instances vary from the different configurations of CPU, memory, storage, and networking capacity [43]. Moreover, users are able to select a mix of resources for special purposes to perform complicated computational tasks. Each instance type can be deployed with different instance sizes, allowing customers to elastically scale the computing resources and balance the target workload. The price of instances varies with different regions in which the users selected and running their instances [44].

4.2.2 Instance Types

A basic comparison between On-Demand instances, Reserved instances and Spot instances, in terms of the unique features, cost and suitable users, is illustrated below [44]:

- On-Demand instances are simply charged by the hours used by users, without any committed terms or upfront payments [44]. On-Demand instances are suitable for: (1) the users who want low cost and facilitate the EC2 resources flexibly and unwilling to pay any long-term commitment fee; (2) the short-term running applications or those with uncertain usage term, but without interruptions; (3) applications under test or development.
- Reserved instances allow the users to pay upfront fee for an instance at the beginning, then use the reserved instances in a relatively cheaper price than On-Demand price, within one or three year term [44]. A significant lower hourly rate for that instance is beneficial in a long-term way. For applications with the need of stable utilisations, AWS announced that using Reserved instances can achieve up to 71% savings compared by using On-Demand instances, in 2013 [44]. It should be noted that, functionally, Reserved instances have the identical performance as On-Demand instances. There are different options for the use of Reserved instances: light, medium and heavy [44]. According to the different upfront fees of these three options, the different hourly usage prices are enjoyed.
- Spot instances allows users to purchase computing capacity with a bid price they provide, without any upfront commitment [44]. Users can enjoy hourly rates which are usually lower than the On-Demand rate. The bid price is considered as the maximum hourly price that users are willing to pay for a particular instance type. The Spot price of AWS EC2 fluctuates based on supply and demand, and varies for different regions and different available instance type in each availability zone [44]. The real-time Spot price of each instance is released to users, which is the lowest feasible price for users to run the Spot instance in that given period. Once the Spot price moves up and goes higher than the customer's bid price, the running instance will be immediately shut down by AWS. When Spot instance running actively, the performance is exactly the same as On-Demand or Reserved instances.

4.2.3 Break-even Analysis of Instances

Table 4.3 shows the upfront fees for different reserved instances in a 1 year reservation usage in 2013. All these numbers are calculated and concluded from the comparison with the cost of a 1 year fully-used On-Demand instance. It can be seen from the table that light used m1.large reserved instances become beneficial after 1981 hours of usage, which occupy 31.7% of yearly usage. For medium and heavy reserved instances, the initial upfront fees are higher than the light reserved instances and they are beneficial much later, after 5029 and 5717 hours respectively. Reserved instances result in a total savings with a lower hourly usage fee of usage although the committed upfront fee paid. A detailed price comparison is given in Table 4.4, including the specific On-Demand price, upfront fee and hourly fee of Reserved instances.

The first plot in Figure 4.1 presents the cost versus hours for these four different instance types, which assume all of them are fully-used for 1 year. The black line 1 is the plot of On-Demand instance; red line 2, green line 3 and blue dot line 4 are light, medium and heavy Reserved instances respectively. The top graph illustrates the results for a 1 year reservation for 1 single instance usage and the bottom one is for the cost of 5 fully used reserved instances. The plots are based on the variant prices of Reserved instances with different committed years, and On-Demand Price from Table 4.4. It shows that, for a single instance, reserved instances will not achieve a lower cost below a certain usage. With more instance used, the reserved instances become economically beneficial, which is compared in Figure 4.1. The top plot reflects the cost versus hours for different instance types of a single instance, but the bottom plot is the usage of 5 instances.

Table 4.3: Fees comparison between Reserved and On-Demand instances

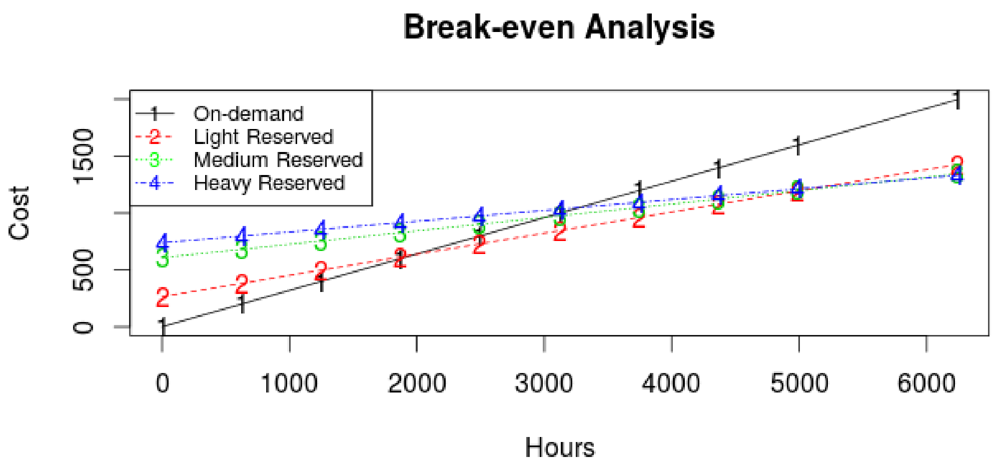
Types (m1.large)	**Most Beneficial After (hours)**	**Runtime of the term**	**Upfront Fee /cascading**	**Usage Fee (R parallel packages)**
On-Demand	0	0%	0	$0.32
Light	1981	31.7%	$266	$0.186
Medium	5029	80.5%	$608	$0.118
Heavy	5717	91.6%	$379	$0.095

4.2.4 StarCluster

StarCluster was initially developed at the Massachusetts Institute of Technology (MIT), by Academics and Researchers (STAR) team [45]. It aims to provide an easy-to-use environment for cluster computing, especially for users and developers on AWS.

TABLE 4.4: Prices for light, medium and heavy Reserved instance

Reserved Instance (m1.large)	**Upfront Fee**	**Hourly Price**	**On-Demand Price**	**Hourly Saving**
Light Utilisation	\$266 in 1 year, \$388 in 3 years	\$0.186 \$0.149	\$0.32	42% 53%
Medium Utilisation	\$608 in 1 year, \$964 in 3 years	\$0.118 \$0.095	\$0.32	63% 70%
Heavy Utilisation	\$739 in 1 year, \$1160 in 3 years	\$ 0.095 \$0.076	\$0.32	70% 76%



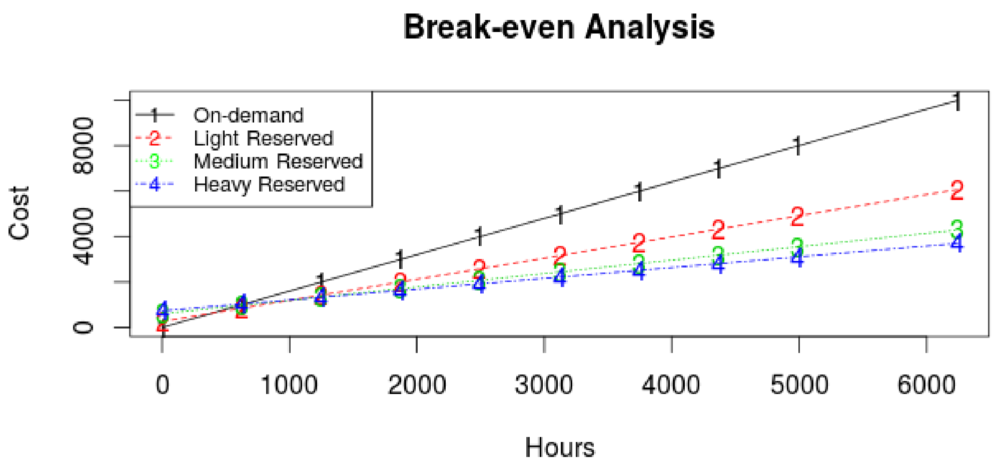


FIGURE 4.1: Break-even plot of AWS Reserved instances and On-Demand instance

StarCluster was chosen as the control toolkit due to its mature support of Hadoop. After creating an AWS account, users are able to execute all the operations using control commands though StarCluster. It not only includes and maintains all the features of AWS cloud service [43] mentioned above, i.e. (1) economical; (2) elastic scaling; (3) no physical presence required; (4) spot instance, but also provide additional benefits [45]:

- Easy configuration. It is easy to request different sets of machines as different requirements using particular configuration plan.
- Avoid software breakage. It is easy to setup the operating system and libraries via custom user AMIs, no need to re-install the software in each start of cluster [45].
- Queueing system. It configures the Oracle Grid Engine queueing system for distributing the computing tasks for the cluster [45], to guarantee that the entire cluster is auto-balanced and not overloaded.
- Passwordless SSH access between nodes. It is useful for administrator to switch and perform the tasks in various nodes across the cluster.
- Dynamically resizing clusters. It supports the elastic control of the cluster in a dynamical way. The users can add and remove nodes from the cluster at anytime through the Oracle Grid Engine [45]. The load balancer of Oracle Grid Engine will add more nodes to relieve the load when the task queue is overloaded.

4.3 Rhipe Package

As introduced in Chapter 2, Rhipe package is an open source R package that allows the users to program the MapReduce jobs in parallel within the R environment [39]. It includes both functions for data analysis and cluster architecture, which enables processing of complex large data.

The concept of Divide and Recombine (D & R) for large-scale data analysis was introduced with the development of Rhipe [40]. From this thesis' point of view, D & R is a superset of MapReduce methodology but with more expandable capacity. In D & R, the input data are usually divided into different subsets. Statistical methods are applied to each of the subsets independently, which means no communication with other datasets [70]. Then the results of each subset are recombined. The concept of D & R consists of three basic computations [40]: S computations are responsible for dividing data into subsets; then the analytic methods are applied to each subset in W computations; the recombination of the outputs from W are processed in B computations, see Figure 4.2. By applying the parallelisation of data in D & R, Rhipe can efficiently processes the large complex data analysis in a scalable manner.

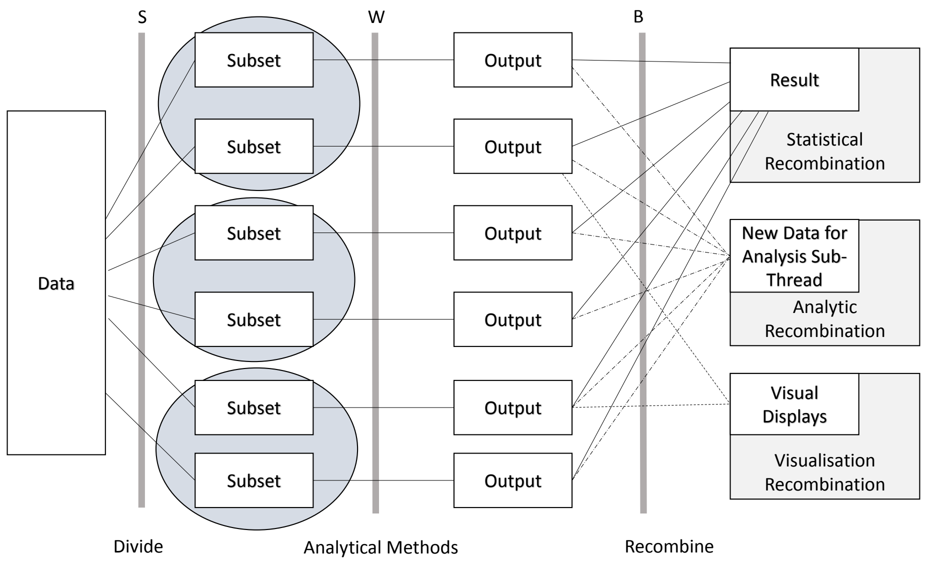


FIGURE 4.2: The D & R computational framework.

4.3.1 Advantages of Rhipe

Data parallel distributed computing methods such as MapReduce have been developed as the tools to deal with large data, but there is few works done about implementing MapReduce in R. There are only few options available to process parallel computing in R. Compared with the other choices, the advantages of using Rhipe in large data are quit remarkable considering its ideal combination of powerful statistical analysis of R and parallel computing in Hadoop. Table 4.5 compares these choices in terms of different aspects:

TABLE 4.5: Advantages of Rhipe

	Rhipe	**Hadoop**	**Pig/Scoobi /cascading**	**Snowfall/ multicore packages**
User Friendly	✓	✗	✓	✓
Handle Large data set	✓	✓	✓	✗
Apply statistical algorithm	✓	✗	✗	✓
Large Data visualisation	✓	✗	✗	✗

Although there are certain merits in Rhipe in large data computing, the drawbacks of Rhipe also need to be highlighted:

1. Need to write R code in map reduce format
2. Need other formal R packages for statistical algorithm computation in large data sets

3. Not as mature and stable as Hadoop, Pig
4. No mature debug process supported by framework
5. Not fault-tolerance in the complicated MapReduce processes

Technically, basing a design on Rhipe as a pure R package [39] limits the developer capacity. However, this thesis is not concerned with the first two issues as the major challenges when using Rhipe. Instead, the powerful statistical ability of R benefits the effectiveness and efficiency of the proposed framework. The basic design of the MapReduce programming model has been systematically introduced in Section 3.2 and how it incorporated with Rhipe is demonstrated through a practical example in Section 4.3.2.1. Since the proposed design involves complex indexing and assembly in MapReduce, the primitive debugging toolset and the relatively low fault-tolerance of Rhipe caused many challenges for implementation and experimentations on large-scale data. However, after solving the issue over logical programming, the advantages of D & R improved the efficiency of time series rolling analysis. Rhipe's visualisation helps developers monitor the status of the complex processing of large data in real-time, Section 4.3.2.2 describes this in detail.

4.3.2 Example

In this section, Rhipe interactions with MapReduce programming model is demonstrated by the way of a toy problem using *iris* dataset. The objective is not to benchmark Rhipe's ability to tackle large datasets, but on how it parallelises the computation. However, it must be noted that Rhipe has a certain overhead while processing any task. For instance, a job on the extremely small set of data can take up to two minutes as a result of overhead from the initialisation and communication between nodes. The results from working with big data, however, is not very sensitive to this overhead and will not be affected by these minor delays. There are more discussions on the performance of big data processing in Chapter 6.

4.3.2.1 Specification of Rhipe

The sample code below demonstrated how MapReduce operates in D & R. Firstly, the *iris* data was partitioned by the class species and stored with a HDFS path. Then, in the MapReduce stage, the split data was systematically assigned with key-value pairs in map and summed up by the Petal.Length in reduce. In the last step, the results were written to HDFS with another output path and the specific MapReduce task configurations were controlled. Map and Reduce are two modes of Hadoop computation, and interact well with D & R computational framework which is illustrated in Figure 4.2. Map is the first parallel computation mode and only computes within each subset. It executes parts of the W computations, as in Figure 4.2. The Reduce mode is able to compute across subsets. Both of B computations and W computations are partially performed in Reduce. The S computations are involved in Map [40].

The following R code snippets outlines how MapReduce model interact with the D & R concept of Rhipe:

Partitioning the Data:

```
path = "/user/lei/demo/testdata
bySpecies <- lapply(unique(iris$Species), function(x) {
    list(as.character(x), subset(iris, Species==x))
})
rhwrite(bySpecies, path)
```

MapReduce Stage:

```
map <- expression({
   lapply(seq_along(map.values),function(r) {
      v <- map.values[[r]]
      k <- map.keys[[r]]
      rhcollect(k, v)
   })
})

re<- expression(
    pre ={ sum <- 0 },
    reduce = { sum <- lapply(reduce.values, function(x) sum(x$Petal.Length)) },
    post = { rhcollect(reduce.key, sum)}
)
```

Results:

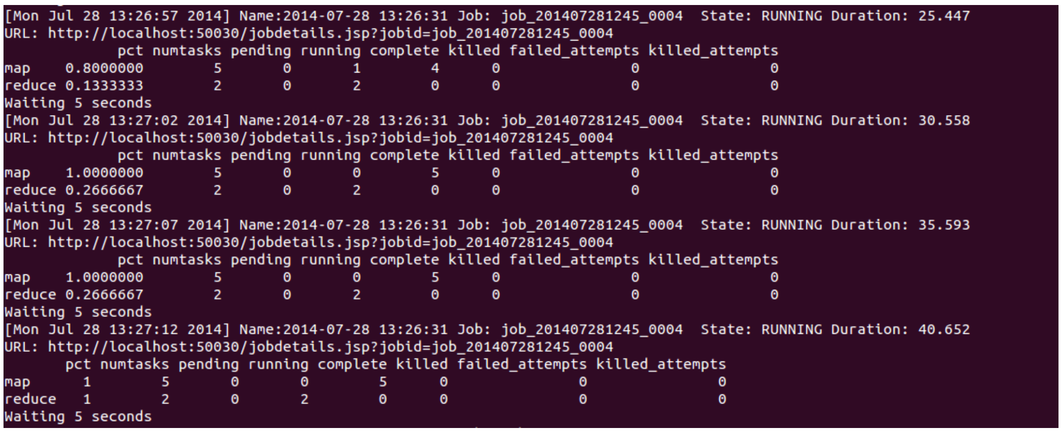
```
  results <- rhwatch(map=map,reduce=reduce,setup=setup,
     input=path,
     output="/user/lei/demo/testre",
     mapred=list(mapred.reduce.tasks=2)
```

Rhipe processes the parallel computations S, W and B with Hadoop [40]. Therefore, it is not necessary to program in Hadoop, or to be deeply familiar with parallel processing. The above elementary computational modes can help developers easily understand and construct their own MapReduce processing in Rhipe. The proposed MapReduce methodology, described with pseudocode in Section 3.2, is applied to this D & R programming model. This skeleton inherits the parallel capacity of MapReduce and has evolved to handle the large-scale time series data in a more scalable and efficient way. The data values in Map, which divided by S computations, are collected by key pairs $(key, value)$ and then are ready to be processed through the analytic methods of W computations. The categorical values from Map are passed to Reduce with re-assigned reduce keys. The analytic methods are applied to the re-grouped reduce values, and then recombined through a further B computation. After B computations, the results not only can be recombined statistically but also can be further analysed to extract the deep statistical characteristics by another analytic recombination [40].

4.3.2.2 Visualisation of Rhipe

Rhipe also provides a detailed visualisation of complex parallel data processing [70]. As each subset contains the data to proceed the MapReduce jobs, it is necessary and important to monitor the processing of D & R computations in real-time. The visualisation of Rhipe starts with the dividing of input data in HDFS, and then the statistical computing on each sampled subset and the final recombination of analytical results in MapReduce. These are visualised either in the forms of R plots or the resource manager interface of Hadoop. The visualisation is a reflection of the D & R computations, as well as a display of the processing status in Hadoop and HDFS [40].

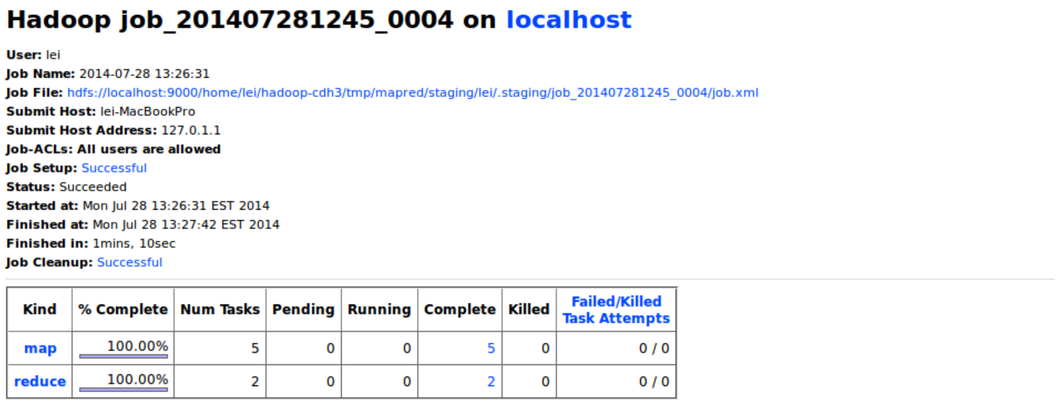
The screenshot Figure 4.3 shows the processing status of a MapReduce job programmed by Rhipe. The specific processing data and the percentage of tasks completion in both the map and reduce stages are included, to reflect the real-time status of the MapReduce jobs. There is an URL link provided which can direct to the Hadoop resource manager web interface. The contents of this URL link are important to the Rhipe's visualisation, which is supported by Apache Hadoop [29]. As Rhipe inherits the advantages of Hadoop, the resource manager web interface is also obtained by Rhipe. It is hosted on the master node of each cluster to monitor the entire MapReduce processing as well as the write and read on HDFS. In Rhipe, it is locally hosted at http://localhost:50030/ or at http://NameNodeHost:50030/ of cloud, where the NameNodeHost should be replaced by the address of master node. There are four key information resources contained in it. The first two resources provide the general information about the current Hadoop job, including the job file name, submission details, and a table of map and reduce tasks



```
[Mon Jul 28 13:26:57 2014] Name:2014-07-28 13:26:31 Job: job_201407281245_0004  State: RUNNING Duration: 25.447
URL: http://localhost:50030/jobdetails.jsp?jobid=job_201407281245_0004
             pct numtasks pending running complete killed failed_attempts killed_attempts
map    0.8000000        5       0       1        4      0               0               0
reduce 0.1333333        2       0       2        0      0               0               0
Waiting 5 seconds
[Mon Jul 28 13:27:02 2014] Name:2014-07-28 13:26:31 Job: job_201407281245_0004  State: RUNNING Duration: 30.558
URL: http://localhost:50030/jobdetails.jsp?jobid=job_201407281245_0004
             pct numtasks pending running complete killed failed_attempts killed_attempts
map    1.0000000        5       0       0        5      0               0               0
reduce 0.2666667        2       0       2        0      0               0               0
Waiting 5 seconds
[Mon Jul 28 13:27:07 2014] Name:2014-07-28 13:26:31 Job: job_201407281245_0004  State: RUNNING Duration: 35.593
URL: http://localhost:50030/jobdetails.jsp?jobid=job_201407281245_0004
             pct numtasks pending running complete killed failed_attempts killed_attempts
map    1.0000000        5       0       0        5      0               0               0
reduce 0.2666667        2       0       2        0      0               0               0
Waiting 5 seconds
[Mon Jul 28 13:27:12 2014] Name:2014-07-28 13:26:31 Job: job_201407281245_0004  State: RUNNING Duration: 40.652
URL: http://localhost:50030/jobdetails.jsp?jobid=job_201407281245_0004
       pct numtasks pending running complete killed failed_attempts killed_attempts
map      1        5       0       0        5      0               0               0
reduce   1        2       0       2        0      0               0               0
Waiting 5 seconds
```

FIGURE 4.3: Rhipe processing example

completion status; the second part contains the detailed information of JobTracker in the forms of job counters, Rhipe timing, RhipeInternal, HDFS counter and Map-Reduce framework; then the last part is comprised of two bar graphs, which illustrate the Map and Reduce completion status visualised. The screenshots of the above four resources can be found in Figure 4.4, Figure 4.5 and Figure 4.6.



Hadoop job_201407281245_0004 on localhost

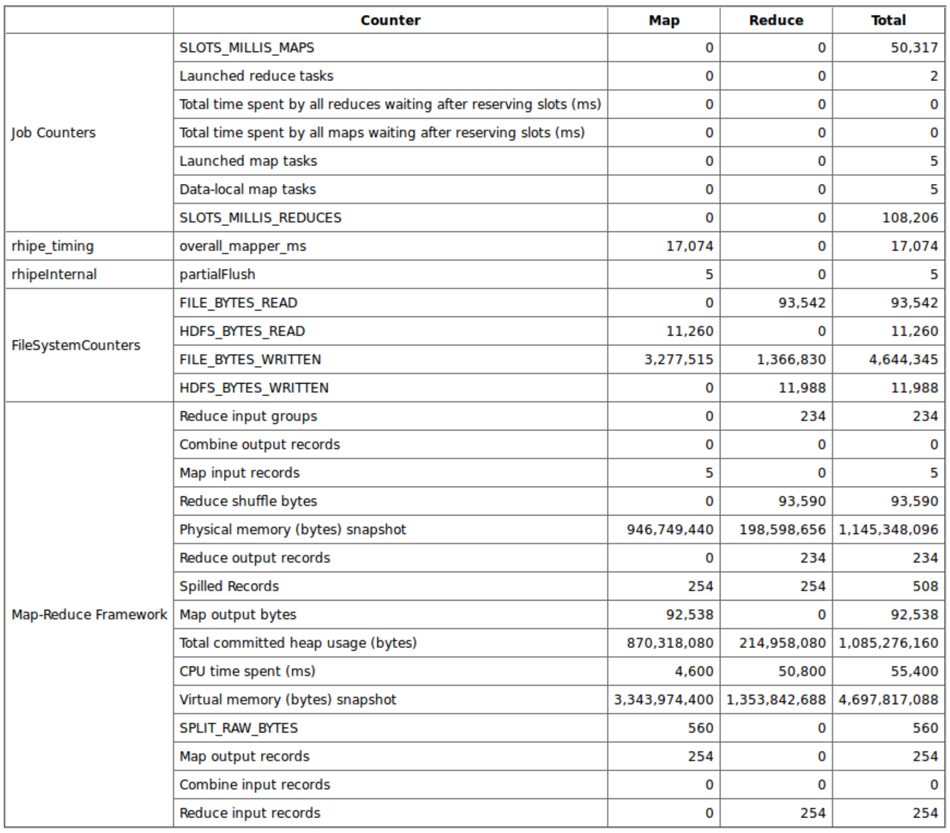
User: lei
Job Name: 2014-07-28 13:26:31
Job File: hdfs://localhost:9000/home/lei/hadoop-cdh3/tmp/mapred/staging/lei/.staging/job_201407281245_0004/job.xml
Submit Host: lei-MacBookPro
Submit Host Address: 127.0.1.1
Job-ACLs: All users are allowed
Job Setup: Successful
Status: Succeeded
Started at: Mon Jul 28 13:26:31 EST 2014
Finished at: Mon Jul 28 13:27:42 EST 2014
Finished in: 1mins, 10sec
Job Cleanup: Successful

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	5	0	0	5	0	0 / 0
reduce	100.00%	2	0	0	2	0	0 / 0

FIGURE 4.4: Example of local Hadoop job

4.4 Summary

This chapter compared two different well-known cloud platforms, AWS and Microsoft Azure, in terms of performance, price and services provided. Then a deep analysis and comparison was conducted on different instance types of AWS, centring on the benefits of different pricing schemes. The next half of the chapter illustrated both advantages and disadvantages of using Rhipe package in relation to the proposed work and how they interact. Finally, a practical coding sample was presented to demonstrate how MapReduce works in Rhipe.



	Counter	Map	Reduce	Total
Job Counters	SLOTS_MILLIS_MAPS	0	0	50,317
	Launched reduce tasks	0	0	2
	Total time spent by all reduces waiting after reserving slots (ms)	0	0	0
	Total time spent by all maps waiting after reserving slots (ms)	0	0	0
	Launched map tasks	0	0	5
	Data-local map tasks	0	0	5
	SLOTS_MILLIS_REDUCES	0	0	108,206
rhipe_timing	overall_mapper_ms	17,074	0	17,074
rhipeInternal	partialFlush	5	0	5
FileSystemCounters	FILE_BYTES_READ	0	93,542	93,542
	HDFS_BYTES_READ	11,260	0	11,260
	FILE_BYTES_WRITTEN	3,277,515	1,366,830	4,644,345
	HDFS_BYTES_WRITTEN	0	11,988	11,988
Map-Reduce Framework	Reduce input groups	0	234	234
	Combine output records	0	0	0
	Map input records	5	0	5
	Reduce shuffle bytes	0	93,590	93,590
	Physical memory (bytes) snapshot	946,749,440	198,598,656	1,145,348,096
	Reduce output records	0	234	234
	Spilled Records	254	254	508
	Map output bytes	92,538	0	92,538
	Total committed heap usage (bytes)	870,318,080	214,958,080	1,085,276,160
	CPU time spent (ms)	4,600	50,800	55,400
	Virtual memory (bytes) snapshot	3,343,974,400	1,353,842,688	4,697,817,088
	SPLIT_RAW_BYTES	560	0	560
	Map output records	254	0	254
	Combine input records	0	0	0
	Reduce input records	0	254	254

FIGURE 4.5: Details of JobTracker

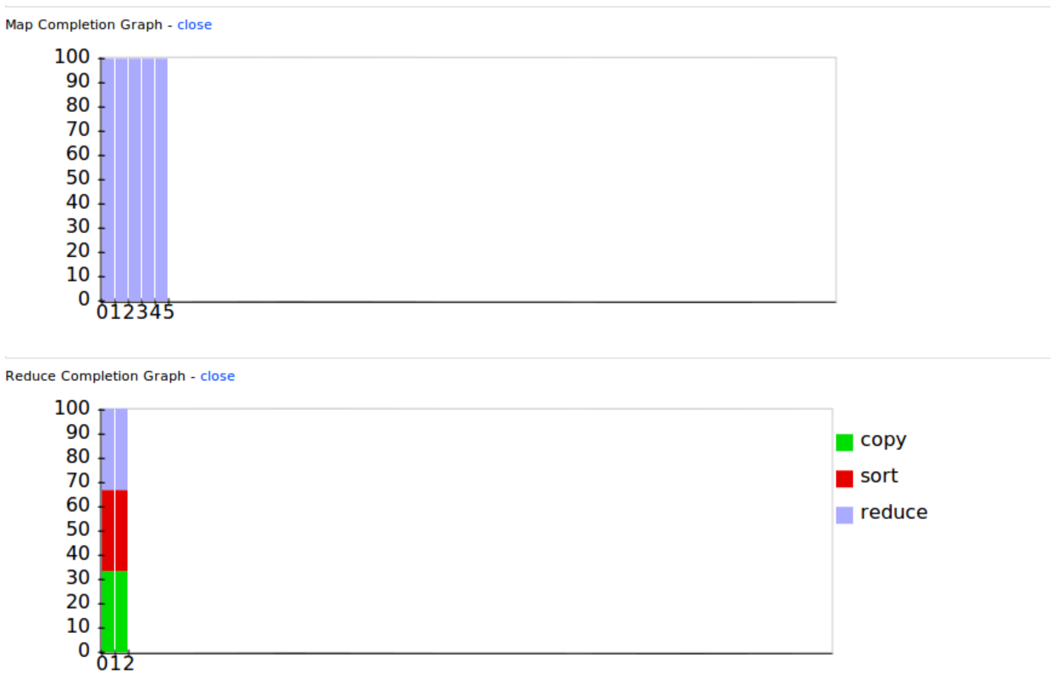


FIGURE 4.6: Completion of map and reduce jobs

Chapter 5

Boosting

In this chapter boosting is discussed in detail. The boosting technique is chosen as an approach to improve the accuracy of the prediction results for the proposed framework. The specific modifications and parameterisation of the boosting algorithm in *gbm* package are studied in this chapter as well.

5.1 Boosting

Boosting was proposed as an ensemble method, to generate a slightly better prediction based on an ensemble of multiple weak hypothesis predictions. It has been proven that various boosting algorithms can successfully improve the prediction quality in a variety of applications [54]. Therefore, boosting techniques have attracted much attention in machine learning and other statistical areas [71–73].

In the late 90s, the application of boosting techniques had a significant progress as Friedman discovered gradient boosting machine [71, 72]. Moreover, he had a further study in linking various boosting algorithms for statistical analysis of estimations [73]. Friedman's study led a new angle of applying boosting methods to areas other than the initial classification area [73]. In the proposals of [74–76], boosting is not only to improve the prediction accuracy but also aimed at estimating the linear and additive models. Currently, boosting is considered more of a regularisation scheme for estimating a model, but still performs competitively in improving predictions [54]. This is the main reason for choosing boosting techniques to further enhance the forecasting of the proposed framework.

However, a series of questions have to be considered before applying boosting algorithms into our real-world predictions: (1) Is boosting suitable to our prediction problem? (2)

Will the prediction be improved through boosting? (3) What is the proper boosting model for the prediction? (4) How to evaluate the boosting results and what is the benchmark?

The rest of this section will elaborate in detail about how boosting works, before moving further to implementation of boosting.

5.1.1 Ensemble Schemes

An ensemble schemes is defined as using a linear combination of multiple base predictions to produce the final prediction [54, 76]. A base procedure is presented as a function $p(\cdot)$ to predict based on input/output pairs $(X_1, Y_1), \cdots, (X_n, Y_n)$ [54]:

$$(X_1, Y_1), \cdots, (X_n, Y_n) \xrightarrow{BaseProcedure} p(\cdot)$$

The base procedures are then combined as an ensemble. In this case, the ensemble of base predictions can be presented as below:

$$\begin{aligned} \text{re-weighted data } M &\xrightarrow{BaseProcedure} p^{[M]}(\cdot) \\ \text{aggregation: } f_A(\cdot) &= \sum_{m=1}^{M} \alpha_m p^{[M]}(\cdot) \end{aligned}$$

The individual data are iteratively assigned weights in each n sample points, namely "re-weighted data". It is assumed that the base procedure allows the estimation of input depending on a weighted sample. The data re-weighting mechanism is equally important as the linear combination coefficients $\alpha_m{}_{m=1}^{M}$ [54]. The variant mechanisms lead to different ensemble schemes with different levels of improvements [74]. Most boosting methods estimate the weights m only based on the previous iterative result $m-1$. The bagging [77] and random forests [78, 79] are the examples of other ensemble schemes.

The idea of applying ensemble methods in boosting was sparked in machine learning area. The AdaBoost algorithm is the most widely used boosting method for binary classification problem [80–82]. More details of AdaBoost algorithm is illustrated in Appendix A.1.

5.1.2 Gradient Boosting Machine

Friedman developed a proposal for boosting algorithms in 1999, which was a general simple framework to produce an optimised boosting results for statistical analysis, namely gradient boosting machine [83].

It is assumed that a regression function $\hat{f}(x)$ can minimise the expectation of a loss function, $\Psi(y, f)$, as shown in equation 5.1 [84].

$$\begin{aligned}\hat{f}(x) &= \underset{f(x)}{\operatorname{argmin}}\, E_{y,x}[\Psi(y, f(x))] \\ &= \underset{f(x)}{\operatorname{argmin}}\, E_x[E_{y|x}[\Psi(y, f(x))|x]]\end{aligned} \tag{5.1}$$

If only focus is on finding the estimates of $f(x)$:

$$\hat{f}(x) = \underset{f(x)}{\operatorname{argmin}}\, E_{y|x}[\Psi(y, f(x))|x] \tag{5.2}$$

It is assumed that $f(x)$ is a function with a finite number of parameters β, based on parametric regression models. The parameters are estimated by selecting the values on (x, y) pairs over the N size training sample, to minimise a loss function [83], e.g. squared error loss. The equation is presented as below:

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{N} \Psi(y_i, f(x_i; \beta)) \tag{5.3}$$

Friedman pointed out that the work is difficult if the function $f(x)$ is estimated non-parametrically [53]. Therefore, the estimate of $f(x)$ should be modified in a greedy function approximation, by using the similar approach in equation 5.1. Assuming that $f_i = f(x_i)$ can decrease the dimension of function.

$$\begin{aligned}J(f) &= \sum_{i=1}^{N} \Psi(y_i, f(x_i)) \\ &= \sum_{i=1}^{N} \Psi(y_i, f_i)\end{aligned} \tag{5.4}$$

The gradient of $J(f)$ implies the local decrease direction in $J(f)$ [83]. Then f is modified as:

$$\hat{f} \leftarrow \hat{f} - \rho \nabla J(f) \tag{5.5}$$

where f is in gradient descent. The parameter ρ is the step size of the decrease direction [83]. However, there are two issues in this step [84]. It only fits f at values of x for those with observations. Another issue is that similar observed values in x generate the similar $f(x)$ values. Friedman found that if using the covariate to approximate the gradient the issues can be solved [73, 83]. His gradient boosting algorithm can be described as below.

Gradient Boosting Algorithm:

Initialise $\hat{f}(x)$ to be a constant, $\hat{f}(x) = \operatorname{argmin}_{\rho} \sum_{i=1}^{N} \Psi(y_i, \rho)$

For t in $1, \ldots, T$ do

1. Compute the negative gradient as the working response

$$z_i = -\frac{\delta}{\delta f(x_i)} \Psi(y_i, f(x_i))|_{f(x_i)=\hat{f}(x_i)} \tag{5.6}$$

2. Fit a regression model, $g(x)$, predicting z_i from the covariates x_i
3. Choose a gradient descent step size as

$$\rho = \underset{\rho}{\operatorname{argmin}} \sum_{i=1}^{N} \Psi(y_i, \hat{f}(x_i) + \rho g(x_i)) \tag{5.7}$$

4. Update the estimate of $f(x)$ as

$$\hat{f}(x) \leftarrow \hat{f}(x) + \rho g(x) \tag{5.8}$$

The boosting algorithm determines the gradient direction at each iteration, where to enhance the fit to the data and select the most suitable model for the prediction [84].

The basic framework suggested above can be extended and improved through various ways. Friedman suggested several choices to develop new boosting algorithms by applying different loss functions and subsampling schemes to improve predictive performance of boosting and reduce its computation time [53, 83]. The *gbm* package applied these modifications as parameters setting of its boosting method. More details are discussed in section 5.2.

5.1.3 Discussion about Boosting Algorithms

Most boosting algorithms are processed as depending on a distribution model to combine the weak predictor algorithms to a final strong predictor. The "weak" learners are assigned with weights in each iteration step, as to estimate the accuracy separately in each learning step of boosting. The data of weak learners are re-weighted. For example, the predictor algorithms with good predictions will add weights and the learners with less accurate prediction lose weights. Different boosting algorithms use different weighting schemes to re-weight the weak predictors iteratively. Overall, the boosting focuses more on the data that weak learners incorrectly predicted in previous iteration.

There are many boosting algorithms. The original ones, proposed by Schapire, were studied and proved that they could not take full advantage of weak learners [51]. Then the term of boosting algorithms were used on the provable algorithms with approximately correct learning formulation [47].

5.1.3.1 How to Choose Weak Learners

Theoretically, boosting is a function that combine weak classifiers to obtain very strong classifier. In the proposition that weak classifiers are slightly better than random on training data, boosting always results in a very strong classifier, even sometimes can eventually provide zero training error. However, it is not absolute that boosting will improve the estimates or predictions.

The choice of weak learners has a pivotal influence on the improvement of accuracy. Then what weak learners should be chosen is a kind of a trade-off between 3 factors [54]

- The bias of the model. A lower bias is almost always better, but one should be careful about something that will overfit as boosting cannot overcome the issue of overfitting.
- The training time for the weak learner. Generally we want to be able to learn a weak learner quickly, as we are going to be building a few hundred or thousand of them.
- The prediction time for the weak learner. If a model has a slow prediction rate, this will lead to a few hundred times slower prediction in the ensemble of them.

5.1.3.2 Boosting Implementation Issues

As illustrated above, there are three key factors for choosing a proper weak learner. However, a question about boosting from a single learner or multiple learners raised the question about input space of boosting. Instead of learning a single weak learner, learn many weak learners that are good at different parts of the input space. The output class selects each base learner with different weights, then will perform better than each single classifier/predictor. In this thesis, the prediction results of MPM were taken as the experimental input for boosting. In order to study how input space and bias of the each base learner's prediction influence the prediction, different combinations of MPM base models with different sizes are compared in Chapter 6 as well. In addition, a rolling window procedure was applied to boosting, marked as "RollingBoosting", which is used to enhance the quality, percision and effectiveness of forecasting. For each rolling window, the training sample size for boosting is selected as 500. Another sequential boosting procedure on a block of data is marked as "BlockBoosting" to compare with the rolling processing. In order to answer the question (2), the error measures described in Chapter 3 were used to evaluate the boosted predictions with the initial results in Chapter 7. For question (3), the appropriate boosting model for the particular experiments was described in Appendix A.4 and demonstrated in in Chapter 7. Question (4) was also answered by the practical experiments in section 6.3.

5.2 Gbm Package

In this section, the R package *gbm* (Generalized Boosted Regression Models) is used to apply the boosting technique for the proposed framework. The *gbm* package extends Freund and Schapire's AdaBoost algorithm and Friedman's gradient boosting machine, which have been discussed above [85]. To answer the question (1) about whether boosting techniques is feasible to improve the prediction for the proposed framework, it is decided to implement the boosting algorithm from *gbm* package for estimation. For the particular problem with many predictor variables, it is expected that boosting can be computationally more efficient, and perform better prediction than the base predictors of MPM.

The implementation and results of boosting can vary form different optimisation schemes, different distribution models, different selection of base learners and loss functions.

5.2.1 Improving Boosting Methods

The boosting methods have the potential to improve the predictive power of "weak" learners [53]. With the development of the *gbm* package, Ridgeway proved that boosting methods can be further optimised to generate more desirable predictions by controlling the learning rate, sub-sampling and a decomposition for interpretation [84]. Given the assumption that the general form of boosting is presented as follows:

$$\hat{f}(x) \leftarrow \hat{f}(x) + E[z(y, \hat{f}(x))|x]. \tag{5.9}$$

The easiest way to obtain the advantages is to substitute the regression procedure for $E_w[z|x]$ [84]. Furthermore, other modifications are discussed in the following subsection.

5.2.1.1 Decreasing Learning Rate

Some researchers claimed that boosting usually hardly overfits when it is additively expanding [84]. However, others believed it is not true [73]. In the iterative learning step of boosting algorithm, a learning rate can be introduced to control the speed of the boosting expansion.

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda E[z(y, \hat{f}(x))|x]. \tag{5.10}$$

The parameter λ controls the speed of gradient step of boosting algorithm. In equation 5.10, it multiples $E[z(y, \hat{f}(x))|x]$ to reduce the error surface of the learning. If $\lambda = 1$ the full gradient steps are the same as equation 5.9. Friedman showed that the learning rate λ controls the regularisation of boosting in 2001 [83] and that λ depends on the iteration number of the steps T. The optimal performance of boosting occurs when setting the learning rate λ as small as possible [84]. The iteration number T can be selected by cross-validation.

5.2.1.2 ANOVA Decomposition

The analysis of variance (ANOVA) is considered as a method to decompose variance into a explainable variance format. For example, the experimental assignments or variances which cannot be explained are usually treated as random error [86]. In the *gbm* package, ANOVA is applied and then then function approximation methods become decomposable [84]. The proposed function can be decomposed as below:

$$f(x) = \sum_j f_j(x_j) + \sum_{jk} f_{jk}(x_j, x_k) + \sum_{jkl} f_{jkl}(x_j + x_k + x_l) + \cdots. \tag{5.11}$$

This function is applied to boosted trees. The stumps of the trees only depend on the first term of 5.11. The second term of 5.11 is the trees with two splits. The order of the approximation in boosting can be easily controlled by the depth of the trees of each iterative step [83]. It is found that the approximation of the additive components of boosting can be well explained through an approximation function, such as generalised additive models, andthe naïve Bayes classifier [84].

5.2.1.3 Relative Influence

Friedman extended the relative influence of a variable for boosting estimation [83]. Taking the tree based example above, the relative influence of the a variable x_j is presented as below:

$$\hat{J}_j^2 = \sum_{\text{splits on } x_j} I_t^2 \tag{5.12}$$

The parameter I_t^2 is produced by splitting the variable x_j, in order to achieve the empirical improvement [84]. In Friedman's proposal in 2002, the boosted models apply the average of the variable's relative influence of variable x_j to all the tress of the boosting algorithm [53].

5.2.2 Loss Function

In the *gbm*, the appropriate selection of distribution model is the key for boosting. The *gbm* package provides several popular functions which can be applied to the boosting algorithm. The Bernoulli or Adaboost model are appropriate for most classification problems. The continuous outcomes can be deployed by selecting the model of Gaussian (for minimising squared error), Laplace (for minimising absolute error), and Guantile Regression (for estimating percentiles of the conditional distribution of the outcome) [85]. The Poisson is able to count outcomes although Gaussian or Laplace are also considered depends on the analytical goals.

5.2.3 Implementing *gbm* Boosting

With all these modifications, the proper boosting methods can be applied by taking the steps below in package *gbm* [84]:

1. Select
   (a) a loss function (distribution)
   (b) the number of iterations, T (n.trees)
   (c) the depth of each tree, K (interaction.depth)
   (d) the shrinkage (or learning rate) parameter, λ (shrinkage)
   (e) the subsampling rate, p (bag.fraction)
2. Initialise $f^*(x)$ to be a constant, $\hat{f}(x) = \underset{\rho}{\operatorname{argmin}} \sum_{i=1}^{N} \Psi(y_i, \rho)$
3. For t in $1, \ldots, T$ do
   (a) Compute the negative gradient as the working response
$$z_i = -\frac{\delta}{\delta f(x_i)} \Psi(y_i, f(x_i))|_{f(x_i)=\hat{f}(x_i)} \tag{5.13}$$
   (b) Randomly select $p \times N$ cases from the dataset
   (c) Fit a regression tree with K terminal nodes, $g(x) = E[z|x]$. This tree is fit using only those randomly selected observations

(d) Compute the optimal terminal node predictions, $\rho_1, \ldots, \rho_K$, as

$$\rho_k = \underset{\rho}{\text{argmin}} \sum_{x_i \in S_k} \Psi(y_i, \hat{f}(x_i) + \rho) \tag{5.14}$$

where S_k is the set of xs that define terminal node k.

(e) Update $\hat{f}(x)$ as

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \rho_{k(x)} \tag{5.15}$$

where $k(x)$ indicates the index of the terminal node into which an observation with features x would fall. Again this step uses only the randomly selected observations.

5.2.4 Example

```
gbm(formula = formula(data),
         distribution = "bernoulli",
         data = list(),
         weights,
         var.monotone = NULL,
         n.trees = 100,
         interaction.depth = 1,
         n.minobsinnode = 10,
         shrinkage = 0.001,
         bag.fraction = 0.5,
         train.fraction = 1.0,
         cv.folds=0,
         keep.data = TRUE,
         verbose = "CV",
         class.stratify.cv=NULL,
         n.cores = NULL)
```

The above example simply illustrate the specific parameters using in *gbm* boosting function. Except the basic formula describing the model to be fit, *distribution* is another critical parameter influencing the performance for choosing the proper distribution model for boosting, which includes "gaussian" for squared error, "laplace" for absolute loss, "bernoulli" for logistic regression for 0-1 outcomes), "multinomial" for classification when there are more than 2 classes, "adaboost" for the AdaBoost exponential loss for 0-1 outcomes and so forth. All these available distribution models are specifically illustrate in Appendix A.3. The *weights* and *n.trees* are the variables that control the weight values used in fitting and the total number of trees, which controls the number of iterations. The *shrinkage* controls the learning rate or step-size reduction applied to each tree in the expansion. The *interaction.depth* is the maximum depth of variable interactions (1 implies an additive model, 2 implies a model with up to 2-way interactions). *cv.folds* controls the number of cross-validation in each boosting processing [85].

Another coding example is given in Appendix A.4 with specific parameter setting used for the experiments in Chapter 7. It further demonstrates how a boosting technique is used for this particular prediction problem.

5.3 Summary

In this chapter, theoretical descriptions of the ensemble scheme of boosting and gradient boosting machine were explained. The boosting implementation issues and how to choose weak learners were discussed as well. The second half of this chapter studied the *gbm* package used for the experiments. Some specific modifications were discussed to improve the boosting method in *gbm*. In the end, a *gbm* boosting example was provided with brief explanation of its parameters.

Chapter 6

Results

This chapter investigates how the proposed framework performs against different cluster sizes. Both synthetic and real-world data are used in the experiments. The efficiency of handling big data is demonstrated in depth, by comparing the proposed design with benchmark and alternative designs. The MPM model is compared to each individual algorithm within MPM. The results show that the architecture of the proposed framework is both efficient and scalable. Boosting is shown to improve the prediction of MPM.

6.1 Experiment Setup

6.1.1 Experiment Environment

All experiments were undertaken on the Amazon Web Service (AWS) Elastic Compute Cloud (EC2) Clusters. The clusters were constructed using the *m1.large* instances type in the US east region. Each node within the cluster has a 64-bit processor with 2 virtual CPUs, 7.5 GiB memory and a 64-bit Ubuntu 13.04 image that included the Cloudera CDH3U5 Hadoop distribution [44]. The statistical computing software R version 3.1.0 [87] and RHIPE version 0.73.1 [39] were installed in each Ubuntu image. RHIPE was used to control the entire MapReduce procedure in the proposed framework.

6.1.2 Test Data

- In order to test the framework, synthetic data was generated from an autoregressive (AR) model using Eq. 3.4. The order $p = 5$ was chosen and ϕ_i were generated randomly. The synthetic time series data starts from 2004-01-01 to 2013-12-31, and is 6.6 MB in size.

- As one of the initial goals of this study, foreign exchange client trade volume data from the Westpac Banking Corporation were used for the MPM and boosting test. The data starts from 2010-05-01 to 2011-05-01.

6.1.3 Preprocessing and Window Size

The preprocessing included normalisation, where the samples were adjusted to have average of $\mu = 0$ and standard deviation $\sigma = 1$. The tests in section 6.2 were performed with a rolling window size $w = 7$, which contained the training data of size $w' = 6$ and the target data with length $h = 1$ for predicting procedure in the framework. Consequently, each predictor model was trained on the previous 3 hours of data to forecast 30 minutes ahead. For the section 6.3, in order to process the data more accurately, we set the window length to a week to predict a 24 hour ahead prediction.

6.2 Performance and Architecture Test

6.2.1 Scaling Test

In order to demonstrate parallel processing efficiency, the SVM prediction method with cross validation was tested using different cluster sizes with 1, 2, 4, 8, 16 and 32 nodes on AWS EC2. The performance of the proposed framework was evaluated by a comparison of execution time and speedup for different cluster sizes.

Table 6.1 shows the execution time for this test. With consideration of the capacity of *m1.large*, the maximum number of map and reduce tasks in each node were limited to the number of CPUs and half the number of CPUs respectively. As can be seen in Table 6.1, scaling is approximately linear up to 32 nodes for the reasonably small example tested. In smaller clusters, the proposed framework scales well. Furthermore, the speedup of 16 and 32 nodes, are 13.09 and 30.83 respectively.

TABLE 6.1: AWS EC2 execution times for scaling test

Cluster Size	**Mappers**	**Reducers**	**Total (sec)**	**SpeedUp**
1 Node	2	1	58514.29	1.00
2 Nodes	4	2	28991.34	2.02
4 Nodes	8	4	13762.74	4.25
8 Nodes	16	8	7092.60	8.25
16 Nodes	32	16	4471.06	13.09
32 Nodes	64	32	1898.15	30.83

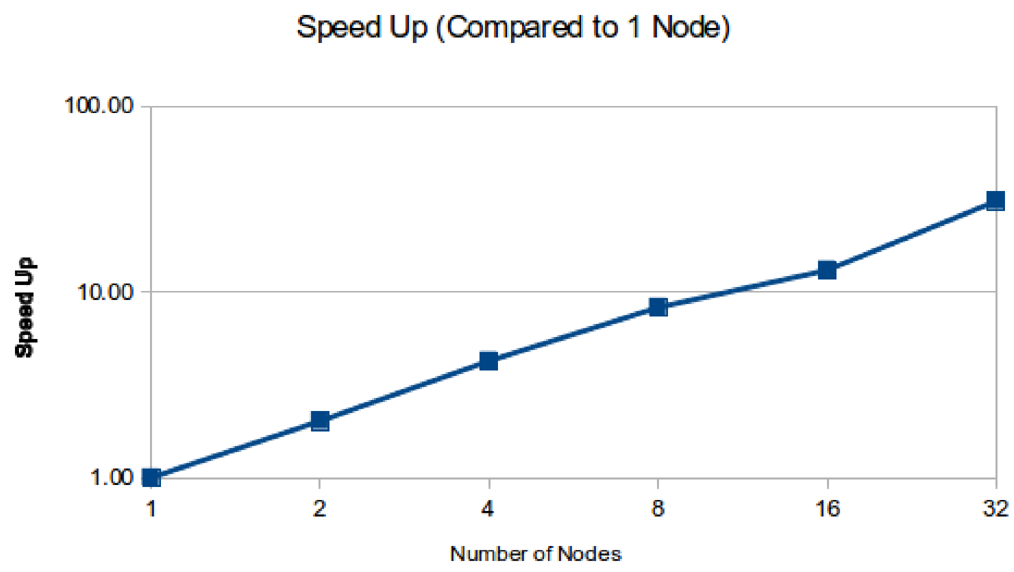


FIGURE 6.1: Speedup of execution time versus cluster size

6.2.2 Cost Estimation

TABLE 6.2: Cost estimation

	1 Node	2 Nodes	4 Nodes	8 Nodes	16 Nodes	32 Nodes
Time						
Exe.time (sec)	58514.29	28991.34	13762.74	7092.60	4471.06	1898.15
Exe.time (hrs)	16.23	8.05	3.82	1.97	1.24	0.52
Counting Hours	17	9	4	2	2	1
Cost						
On-Demand Upfront cost: \$0 Hourly price: \$0.32	5.44	5.76	5.12	5.12	10.24	10.24
L-Reserved 1year Upfront cost: \$266 Hourly price: \$0.186	3.162	3.348	2.976	2.976	5.952	5.952
L-Reserved 3year Upfront cost: \$388 Hourly price: \$0.149	2.465	2.610	2.320	2.320	4.640	4.640
M-Reserved 1year Upfront cost: \$608 Hourly price: \$0.118	2.006	2.124	1.888	1.888	3.776	3.776
M-Reserved 3year Upfront cost: \$964 Hourly price: \$0.095	1.615	1.710	1.520	1.520	3.040	3.040
H-Reserved 1year Upfront cost: \$739 Hourly price: \$0.095	1.615	1.710	1.520	1.520	3.040	3.040
H-Reserved 3year Upfront cost: \$1160 Hourly price: \$0.076	1.292	1.368	1.216	1.216	2.432	2.432

Table 6.2 provides a detailed cost estimation comparing different AWS EC2 instance

types. All these numbers are calculated using the price of the instance *m1.large* reported in Section 4.2.3 from Table 4.4. In the table the upfront fee and hourly price are listed under the name of each instance type respectively. Two major assumptions were made by AWS to calculate the cost: (1) the pricing of AWS is the per instance-hour consumed for each instance, from the time an instance is launched until it is terminated or stopped. Therefore, total cost t is equal to the total working hours h times instance numbers i times the hourly price p; and (2) each partial instance-hour consumed is billed as a full hour. For example, 16.23 executing hours will be counted as 17 hours.

It can be seen from the table the 4-node and 8-node clusters achieve the minimal cost with the best execution time. The estimated costs of light, medium and heavy Reserved instances are significantly lower than On-Demand instances. However, the different upfront fees with a different committed usage period of Reserved instances have to be taken into account while considering the total budget (see Table 4.4). Other circumstances, namely project life span and project usage requirements, also need to be considered when selecting the optimal payment strategy.

6.2.3 Multi Predictor Model Test

The time series prediction algorithms described in the previous section: (1) Support Vector Machines (SVM), (2) Autoregressive Integrated Moving Average (ARIMA), (3) Artificial Neutral Network (NN), (4) Naïve (ARIMA(0,1,0)) and (5) Exponential Smoothing State Space (ETS) model were tested on the synthetic data set individually.

In addition, we used a multi-predictor model (MPM) scheme to improve the efficiency by batching predictors in the proposed framework. In this scheme, a *multi-predictor* function is called in the Map or the Reduce, which in turn applies all user supplied predictors to each data window, returning a vector of prediction results (and error measures) for every predictor. Following the MapReduce, MPM selects the best predictor for the tested/selected time series by using an error measure (RMSE and MAPE) for each prediction model. We used R's forecast package [3] for ARIMA, ANN, Naïve and ETS models, and e1071 package [88] for training SVM models. AIC was used to chose model parameters in ARIMA and ETS models, while SVM and NN models were cross-validated as outlined in Section 3.4.3.

Table 6.3 compares MPM with individual prediction algorithms in terms of execution time, RMSE and MAPE. All tests were undertaken in an AWS EC2 cluster with 16 nodes. Relative execution time (RET) is calculated as a ratio of the model execution time compared to the MPM execution time. As expected, simpler models execute faster.

Table 6.3: Performance comparison of different predictor model on a 16 node cluster.

Predictor model	**Execution time (s)**	**RET**	**RMSE**	**MAPE**
SVM	2935.77	0.67	3.372	0.508
ARIMA	1729.50	0.40	3.445	0.545
NN	1521.24	0.35	4.641	0.755
Naïve	1476.72	0.34	2.381	0.419
ETS	1585.19	0.36	3.449	0.545
MPM	4351.05	1	2.381	0.419

SVM, which is cross-validated several times, takes longer, and MPM, which runs all predictor models, takes the longest. Despite this, MPM is 2.1× faster than executing all models individually, as the former requires significantly less I/O than the latter. The last two columns show the error associated with each model's prediction. For MPM, the minimum RMSE and MAPE obtained from the best-fitted prediction model are presented. The results indicate that for the particular synthetic AR test data generated, the Naïve model has the lowest prediction error, which is also reported by the MPM.

6.2.4 Data Split Handling Comparison

In the proposed framework, with the help of index pool, a complex data split algorithm is designed to balance the execution of prediction methods between Map and Reduce. In order to evaluate the efficiency of the proposed design, a *straightforward* design as discussed in Section 3.3.4, was implemented without this algorithm.

Table 6.4: Computational efficiency of the proposed architecture.

Cluster size	**Proposed framework (s)**	**Straightforward framework (s)**	**Improvement %**
16 Nodes	4351.05	5359.92	23%
32 Nodes	2444.17	3566.94	45%

Table 6.4 compares the proposed framework with the *straightforward* benchmark framework for the same MPM prediction model, elaborately demonstrating the performance of both frameworks for two different cluster sizes. The results clearly show that the proposed framework is more efficient than the benchmark framework. In Section 3.3.4, there is a detailed conceptual discussion illustrating the architecture design, respect to MapReduce programming model. The experimental results further showed that instead of applying all prediction models in Reduce stage, systematic/reasonable processing of

assembling data splits and employing prediction procedure in both Map and Reduce procedures is more efficient. The significant improvements in execution time are due to the reduced overhead of data being cached before being moved from the Map to the Reduce, and then lead to a significant reduction in communication time.

6.3 Boosting Result Comparison

In order to improve the prediction results, a boosting technique was applied to the collection of MPM prediction results. The test data used is the client volume data of Westpac from 2010-05-01 to 2010-06-01 in four currencies, USD, EUR, NZD and JPY. The prediction results from MPM (which includes ELM, Naïve, ETS, ARIMA and NN) were generated through a 50 hours rolling window MapReduce processing.

The **benchmark** setup for boosting method is the ARIMA(0,0,0) model, namely Zero model. In this context, it is the long term average of Westpac client volume data.

The boosting algorithm selected is from the *gbm* R package. As this procedure is time-consuming, it is processed as a post-processing step after the rolling window time series prediction. It is expected to produce a more accurate prediction from the boosting of all “weak” learners’ forecasting results.

6.3.1 HitRate

In addition to the error measures (RMSE and SMAPE) mentioned previously, HitRate (abbreviated as HR in results) is also used the quantity the prediction improvements after boosting. HitRate is traditionally used as a measure of business performance associated with sales. There are only two basic operations in currency trading, sell or buy. The positive numbers in results stand for the volume bought in by clients. In contrast, the numbers of currency volume sold out by clients are negative. The other two error measures, RMSE and SMAPE, are able to reflect the size of error between the prediction and actual data, but not the direction of the trades (negative or positive). Except the measures of showing the the magnitude of the volume clients are willing to trade, another indicator is needed to prove that the predictions are not just arbitrary numbers. Consequently, HitRate is being used in this section, to help on a further measurement of the bias precision of predictive results. It can benchmark the prediction by calculating the accuracy on the predictive numbers’ sign [89]. HitRate generally reflects the possible direction of client trading in a quantitative way.

6.3.2 Boosting Sample

TABLE 6.5: Boosting results for client volume prediction

Prediction Method	**RMSE**	**Ratio**
ELM	9116976.061	1.003
Naïve	12347157.882	1.359
ETS	9793595.494	1.078
ARIMA	9119264.403	1.003
NN	10424333.874	1.147
Zero	9088322.466	1
BlockBoosting	9049643.819	0.996
RollingBoosting	8612966.691	0.948

Table 6.5 compares the prediction results achieved from MPM with the boosting results in post-processing. As shown in the table, both BlockBoosting and Rollingboosting results are better than the others, compared to the benchmark Zero model. In order to have a clear comparison about the RMSE of each model, "Ratio" is calculated by taking the RMSE of Zero model as the base. It indicates that no primary prediction model of MPM performed better than the benchmark Zero as listed in the above table. However, the boosting post-processing improved the prediction based on the collection of the former results, achieving 1% and 5% better in BlockBoosting and RollingBoosting perspectively.

6.3.3 Boosting Test

Table 6.6 includes the boosting results of client volume the numbers in four different currencies, USD, EUR, NZD and JPY. It contains all of prediction results from all five "weak" learners (ELM, Naïve, ETS, ARIMA and NN) with the two boosting results (BlockBoosting and RollingBoosting) and benchmark Zero model of ARIMA(0,0,0). Among the primary "weak" prediction algorithms, ELM performs the best in terms of RMSE in all different currencies, by considering RMSE as the most accurate error measure as the standard. As expected after boosting, the boosted prediction results are slightly better than all primary models, even ELM. This can be clearly seen by the ratio of "vsZero", which is a number by comparing the difference of RMSE between each model with the benchmark Zero Model. The "vsZero" of rolling boosting results in USD, EUR, NZD and JPY, are 0.3%, 0.32%, 0.21% and 0.2% respectively, which are all the smallest numbers within each currency. Although it indicated that the boosted results are still not better than the Zero model, the forecasting results are improved through boosting, compared to the primary predictive results of "weak" models from MPM. The random nature of the financial time series results in the low HitRate of the predictions. In summary, boosting is feasible and able to improve the accuracy of time series prediction.

Table 6.6: Boosting prediction results in four different foreign currencies

Prediction Method	**RMSE**	**SMAPE**	**HR**	**vsZero**
USD				
ELM	7811469	162.12	52.28	0.93
Naïve	10526768	120.98	52.56	36.01
ETS	8549324	165.25	52.24	10.46
ARIMA	7825735	162.3	51.67	1.11
NN	50102417	167.34	53.88	547.33
Zero	7739856	100	0	0
BlockBoosting	7491540	167.12	31.26	-3.21
RollingBoosting	7762793	162.23	53.36	0.3
EUR				
ELM	2305307	180.52	57.95	0.73
Naïve	3210685	131.64	48.98	40.29
ETS	2684418	181.19	60.78	17.3
ARIMA	2308534	180.77	57.49	0.87
NN	5629850	183.71	53.12	146
Zero	2288558	100	0	0
BlockBoosting	2302224	183.09	83.09	0.6
RollingBoosting	2295874	182.02	52.78	0.32
NZD				
ELM	4239166	186.5	49.8	0.92
Naïve	6024310	139.74	30.57	43.41
ETS	4459842	186.62	54.18	6.17
ARIMA	4244371	186.83	49.18	1.04
NN	7434540	189.02	49.34	76.99
Zero	4200645	100	0	0
BlockBoosting	4202826	189.56	25.65	0.05
RollingBoosting	4209303	187.32	57.86	0.21
JPY				
ELM	233010856	189.57	49.95	0.91
Naïve	327881977	143.47	21.35	41.99
ETS	248021897	189.75	53.42	7.41
ARIMA	233448633	162.3	48.72	1.1
NN	434660164	167.34	52.09	88.23
Zero	230914656	100	0	0
BlockBoosting	232134345	167.12	25.54	0.53
RollingBoosting	231370833	162.23	49.85	0.2

Table 6.7: Boosting results of different combination of models

Models Number	**Prediction Methods**	**RMSE**	**SMAPE**	**HR**	**vsZero**
USD					
5 Models	ELM, Naïve, ETS, ARIMA, NN	7762793	162.23	**53.36**	0.3
3 Models	ELM, Naïve, ETS	7781564	162.55	52.75	0.54
3 Models	ELM, ETS, ARIMA	**7756277**	**162.18**	52.15	**0.21**
EUR					
5 Models	ELM, Naïve, ETS, ARIMA, NN	2295874	182.02	52.78	0.32
3 Models	ELM, Naïve, ETS	2294882	181.93	53.63	0.28
3 Models	ELM, ETS, ARIMA	**2294315**	**181.74**	**53.97**	**0.25**
NZD					
5 Models	ELM, Naïve, ETS, ARIMA, NN	4209303	187.32	57.86	0.21
3 Models	ELM, Naïve, ETS	4207867	187.14	57.47	0.17
3 Models	ELM, ETS, ARIMA	**4205274**	**186.96**	**58.72**	**0.11**
JPY					
5 Models	ELM, Naïve, ETS, ARIMA, NN	231370833	190.72	49.85	0.2
3 Models	ELM, Naïve, ETS	**231317301**	190.75	49.64	**0.17**
3 Models	ELM, ETS, ARIMA	231320119	**190.68**	**50.66**	0.18

6.3.4 Measuring Effectiveness of Learner Combinations

Table 6.7 compares the boosting results of different model combinations in all four currencies, USD, EUR, NZD and JPY. It is clearly seen that selecting more learners for boosting did not result in better prediction. In the table, the predictive results of the 3 model combination are mostly better the 5 model results, in terms of RMSE and SMAPE. The only reasonable explanation for this is due to the particular extreme "weak" learners (NN for example) worsening the boosting prediction significantly. Furthermore, the different combination of models is another factor influencing predictive improvement. As the bias varied from different "weak" learners, the different selection of models led to different total bias as well. Table 6.6 in previous session clearly presented, ELM, ETS and ARIMA constantly generated better predictions than the other two models, Naïve and NN. Consequently, for most cases, the predictions from the "good" combination of ELM, ETS and ARIMA are slightly better than the results of "weak" combination of ELM, Naïve and ETS. Due to the random nature of the foreign exchange data used for experiments, there are slight variances among the results, for example, in JPY, the "good" combination of predictive models performed slightly worse than the "weak" one in terms of RMSE.

6.4 Summary

The results from the scaling test and the multi-predictor model test showed that the proposed methodology can efficiently handle the rolling window time series problem using the MapReduce programming model. The feasibility of the proposed architecture design was further demonstrated in the data split handling test compared to a straightforward approach. A detailed cost estimation was summarised using different AWS instance types. The boosting results from four currencies indicated that the boosting technique is feasible for improving the accuracy of prediction. The different combinations of learning models lead to the different degrees of boosting improvements.

Chapter 7

Conclusion

7.1 Summary

In this thesis, we have presented an efficient framework for rolling window time series prediction, with a focus on computational efficiency for large-scale data.

Inspired by the idea of applying R-based forecast algorithms in a parallel fashion, the problems of implementing rolling time series prediction were explained. The distinguishing feature of time series, namely strong dependency between the observations, prevent straightforward parallelisation. The new index pool technique developed in this work allows incomplete windows across data splits be to easily reassembled via the sorting stage of MapReduce. Additionally, our approach allows users to only be concerned about designing prediction algorithms, with data management and parallelisation being handled by the framework. The MPM technique contains a series of prediction models and, after processing, can automatically return the best prediction result with a comprehensive comparison on predictive errors. With the help of AWS cloud computing, the proposed work was applied to cloud clusters and able to resolve the large scale rolling time series prediction. The proposed framework has been evaluated in terms of scaling performance, architecture efficiency and predictor auto-selection. The cost of running the proposed work is also estimated using different AWS instance types.

Within the platform, boosting was added as a post processing to improve the prediction after the MapReduce processing of rolling predictions. We showed that boosting technique is feasible to further increase prediction accuracy. A further study of selecting the optimal combination of learning models was presented.

7.2 Further Directions

First of all, it would be interesting to extend the current platform into a generalised R library, to allow easy deployment of automatic time series rolling window predictions. Also it is possible to wrap more algorithms to extend its capability of the MPM of the proposed platform. Both customised prediction algorithms and existing R forecasting packages are considered.

A second interesting direction is to exploit the present work in other environments. Currently, all technical work is limited to R environment. However, the theoretical architecture and design can be generalised in such a way that it is portable and other programming languages can be targeted. This could possibly be targeted with other more sophisticated platforms or applications but need more research.

The last consideration is to extend the proposed architecture design to not only time series forecasting but other analyses, such as classification and anomaly detection. As the unique dependency feature exists in any types of time series, any research related to time series analysis is facing the difficulty of splitting the time series data in an appropriate way and process in parallel. The notion of indexing pool can be beneficial in other areas to handle the time series data split issue without data loss.

Appendix A

Boosting Example

A.1 AdaBoost

The AdaBoost algorithm is the most well known boosting algorithm used in numerous areas, firstly developed in 1995 by Freund and Schapire [82]. The application of AdaBoost focus mainly on the binary valued weak predictions or classification, which is distinguished from the other boosting methods [90].

A.1.1 AdaBoost Algorithm

AdaBoost Algorithm Pseudocode:

```
Give: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X$,$y_i \in \{-1, +1\}$
Initialise $D_1 = 1/m$ for $i = 1, \ldots, m$
For $t = 1, \ldots, T$ :
    • Train weak learner using distribution $D_t$
    • Get weak hypothesis $h_t : X \to \{-1, +1\}$
    • Select $h_t$ with the low weight error:
          $\epsilon_t = \Pr_{i \sim D_t}[h_t(x_i) \neq y_i]$.
    • Choose $\alpha_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t})$
```

- Update:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if} h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if} h_t(x_i) \neq y_i \end{cases}$$
$$= \frac{D_t(i)\exp(-\alpha_t y_t h_t(x_i))}{Z_t}$$

where Z_t is a normalisation factor (chose so that D_{t+1} will be a distribution).

- Output the final hypothesis:

$$H(x) = \text{sign}(\textstyle\sum_{t=1}^{T} \alpha_t h_t(x))$$

Pseudocode for AdaBoost algorithm is shown as above [91]. There are m training sets $(x_1, y_1), \ldots, (x_m, y_m)$ as the input. Each value of x_i is in the domain X. The prediction values are binary $y_i \in \{-1, +1\}$. Adaboost trains the weak learners iteratively in the rounds of $t = 1, \ldots, T$. The distribution D_t is used by each training sample. The aim of learning procedure is to select the weak learner with low weighted error ϵ_t relative to D_t, in order to reduce the training error. The $D_t(i)$ denoted the weight of the distribution D_t on example i in round t. The final combined hypothesis H is the sign of the aggregation of weighted weak hypotheses.

A.2 Training and Weighting of Boosting

Training

The learning from weighted data of boosting contains two major procedures: training and weighting [54]. Assuming the basic boost classifier can be presented in the form of:

$$F_T(x) = \sum_{t=1}^{T} f_t(x)$$

where the f_t is the weak learner with input value x aims to return the classification result. The output of the weak learners are the predicted object class. The real value identifies the confidence level of the predicted classification [91]. The weak hypothesis $h(x_i)$ is generated by each weak learner on each training sample [54]. At each round of t, a weak learner is iteratively selected and with the coefficient α_t as below:

$$E_t = \textstyle\sum_i E[F_{t-1}(x_i) + \alpha_t h(x_i)]$$

The training errors of each iteration of t are summed E_t with error function $E(F)$. The $F_{t-1}(x)$ denotes the boost predictor/classifier from the previous iteration of training.

Weighting

The weighting scheme of boosting is responsible to assign the weights to each hypothesis results at each iterative training process. The weights assigned is used to distinguish and select the weak learners in each iteration step of boosting [81]. For good classification results weights are added to the particular classifier and the relative worse classifier will lose weights. The iteration process would be stopped at round m to avoid overfitting. The cross-validation can be applied for tuning parameter [54].

A.3 Distribution Models of *gbm*

This section lists all the mathematical details for distribution model options offered in the *gbm* package. In each model, the property of deviance, initial value, the gradient and the terminal node estimates are presented in detail [84].

A.3.1 Gaussian

Deviance $\frac{1}{\sum w_i} \sum w_i(y_i - f(x_i))^2$
Initial value $f(x) = \frac{\sum w_i(y_i - o_i)}{\sum w_i}$
Gradient $z_i = y_i - f(x_i)$
Terminal node estimates $\frac{\sum w_i(y_i - f(x_i))}{\sum w_i}$

A.3.2 AdaBoost

Deviance $\frac{1}{\sum w_i} \sum w_i \exp(-(2y_i - 1)f(x_i))$
Initial value $\frac{1}{2} \log \frac{\sum y_i w_i e^{-o_i}}{\sum (1-y_i) w_i e^{o_i}}$
Gradient $z_i = -(2y_i - 1)\exp(-(2y_i - 1)f(x_i))$
Terminal node estimates $\frac{\sum (2y_i - 1) w_i \exp(-(2y_i - 1)f(x_i))}{\sum w_i \exp(-(2y_i - 1)f(x_i))}$

A.3.3 Bernoulli

Deviance $-2\frac{1}{\sum w_i} \sum w_i(y_i f(x_i) - \log(1 + \exp(f(x_i))))$
Initial value $\log \frac{\sum w_i y_i}{\sum w_i(1-y_i)}$
Gradient $z_i = y_i - \frac{1}{1+\exp(-f(x_i))}$
Terminal node estimates $\frac{\sum w_i(y_i - p_i)}{\sum w_i p_i(1-p_i)}$ where $p_i = \frac{1}{1+exp(-f(x_i))}$
Notes:

- For non-zero offset terms, the computation of the initial value requires Newton-Raphson. Initialise $f_0 = 0$ and iterate $f_0 \leftarrow f_0 + \frac{\sum w_i(y_i - p_i)}{\sum w_i p_i(1-p_i}$ where $p_i = \frac{1}{1+\exp(-(o_i+f_0))}$

A.3.4 Laplace

Deviance $\frac{1}{\sum w_i} \sum w_i |y_i - f(x_i)|$
Initial value $median_w(y)$
Gradient $z_i = sign(y_i - f(x_i))$
Terminal node estimates $median_w(z)$
Notes:

- $median_w(y)$ is the weighted median, as the solution to the equation $\frac{\sum w_i I(y_i \leq m)}{\sum w_i} = \frac{1}{2}$
- but currently it is not supported in *gbm*.

A.3.5 Quantile regression

Deviance $\frac{1}{\sum w_i}(\alpha \sum_{y_i > f(x_i)} w_i(y_i - f(x_i)) + (1-\alpha)\sum_{y_i \leq f(x_i)} w_i(f(x_i) - y_i))$
Initial value $quantile_w^{(\alpha)}(y)$
Gradient $z_i = \alpha I(y_i > f(x_i)) - (1-\alpha) I(y_i \leq f(x_i))$
Terminal node estimates $quantile_w^{(\alpha)}(z)$
Notes:

- $quantile_w^{(\alpha)}(y)$ is the weighted quantile, as the solution to the equation $\frac{\sum w_i I(y_i \leq q)}{\sum w_i} = \alpha$

A.3.6 Cox Proportional Hazard

Deviance $-2 \sum w_i(\delta_i(f(x_i) - \log(R_i/w_i)))$
Initial value 0 Gradient $z_i = \delta_i - sum_j \delta_j \frac{w_j I(t_i \geq t_j) e^{f(x_i)}}{\sum_k w_k I(t_k \geq t_j) e^{f(x_k)}}$
Terminal node estimates Newton-Raphson algorithm

1. Initialise the terminal node predictions to 0, $\rho = 0$
2. Let $p_i^{(k)} = \frac{\sum_j (k(j)=k) I(t_j \geq t_i) e^{f(x_i)+p_k}}{\sum_j I(t_j \geq t_i) e^{f(x_i)+p_k}}$
3. Let $g_k = \sum w_i \alpha_i (I(k(i) = k) - p_i^{(k)})$

4. Let H be a $k \times k$ matrix with diagonal elements
   (a) Set diagonal elements $H_{mm} = \sum w_i \alpha_i p_i^{(m)}(1 - p_i^{(m)})$
   (b) Set off diagonal elements $H_{mm} = -\sum w_i \alpha_i p_i^{(m)} p_i^{(n)}$
5. Newton-Raphson update $\rho \leftarrow \rho - H^{-1} g$
6. Return to step 2 until convergence

Notes:

- t_i is the survival time and δ_i is the death indicator.
- R_i is the hazard for the risk set, $R_i = \sum_{j=1}^{N} w_i I(t_j \geq t_i) e^{f(x_i)}$.
- $k(i)$ is the indexes of the terminal node in observation i.

A.3.7 Poisson

Deviance $-2\frac{1}{\sum w_i} \sum w_i (y_i f(x_i) - \exp(f(x_i)))$
Initial value $f(x) = \log(\frac{\sum w_i y_i}{\sum w_i e^{o_i}})$
Gradient $z_i = y_i - \exp(f(x_i))$
Terminal node estimates $\log \frac{\sum w_i y_i}{\sum w_i \exp(f(x_i))}$

A.4 Boosting Example

```
doBoost<-function(datatrain,datatest) {
      gbm1 <- gbm(Y~X1+X2+X3+X4+X5, # formula
      data=datatrain, # train dataset
      var.monotone=c(1,1,1,1,1), # +1 monotone increase
      distribution="gaussian", # gaussian model
      n.trees=10, # number of trees
      shrinkage=0.001, # shrinkage (learning rate),
      interaction.depth=1, # 1: additive model
      bag.fraction = 0.5, # subsampling fraction
      train.fraction = 1, # fraction of data for training
      n.minobsinnode = 10, # minimum total weight needed in each node
      cv.folds = 3, # do 3-fold cross-validation
      keep.data=TRUE, # keep a copy of the dataset with the object
```

```
        verbose=FALSE, # don’t print out progress
        n.cores=1)

        best.iter <- gbm.perf(gbm1,method="cv")#method="OOB")
        f.predict <- predict(gbm1,datatest,best.iter)
        f.predict <- as.numeric(f.predict)
        return(f.predict)
}
```

The above example gives the details of the *gbm* boosting technique with practical parameter settings. By formula, it takes the predictions from five basic models of MPM as the weak prediction hypothesis, to generate the final boosting prediction. The “gaussian” distribution model is selected as the most appropriate choice for the case. A slow learning rate 0.001 and 10 trees are set to the control the boosting learning rate and iteration numbers, respectively. The weighting mechanism is selected as increasing 1 weight for the selected learner in each iterative step. The additive model is used for the maximum depth of variable interactions. Each boosting process is under 3-fold cross-validation.

Bibliography

[1] W.A. Fuller. *Introduction to Statistical Time Series*. Wiley Series in Probability and Statistics. Wiley, 1996. ISBN 9780471552390.

[2] R.S. Tsay. *Analysis of Financial Time Series*. CourseSmart. Wiley, 2010. ISBN 9781118017098.

[3] Rob J Hyndman with contributions from George Athanasopoulos, Slava Razbash, Drew Schmidt, Zhenyu Zhou, Yousaf Khan, Christoph Bergmeir, and Earo Wang. *forecast: Forecasting Functions for Time Series and Linear Models*, 2014. URL `http://CRAN.R-project.org/package=forecast`. R package version 5.3.

[4] Rob J. Hyndman and Yeasmin Khandakar. Automatic Time Series Forecasting: The forecast Package for R. *Journal of Statistical Software*, 27(3), July 2008.

[5] The R Project for Statistical Computing. URL `http://www.r-project.org/`.

[6] Ronald C Taylor. An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics. In *Proceedings of the 11th Annual Bioinformatics Open Source Conference (BOSC)*, 2010.

[7] IBM Software. What is Hadoop? URL `www.ibm.com/software/data/infosphere/hadoop/`.

[8] George E. P. Box, Gwilym M. Jenkins, and Gregory C. Reinsel. *Time Series Analysis: Forecasting and Control*. Wiley Series in Probability and Statistics, 4th edition, June 30, 2008.

[9] Lei Li, Farzad Noorian, Duncan J.M. Moss, and Philip H.W. Leong. Rolling Window Time Series Prediction using Mapreduce. *In Proc. IEEE International Conference on Information Reuse and Integration (IRI)*, pages 757–764, August 2014.

[10] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. Searching and Mining Trillions of Time Series Subsequences under Dynamic Time Warping. In

Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 262–270, 2012.

[11] Rob J. Hyndman and George Athanasopoulos. *Forecasting: Principles and Practice*. Otexts, 2013. URL https://www.otexts.org/fpp.

[12] Wayne A Fuller. *Introduction to Statistical Time Series*, volume 428. John Wiley & Sons, 2009.

[13] Jessica Utts. *Seeing Through Statistics*. Cengage Learning, 3rd edition, 2004.

[14] Shanta Rangaswamy, Shobha G., Samir Sherif, Satvik Nelakant, and Vaishakh B N. Time Series Data Mining Tool. *International Journal of Research in Computer and Communication Technology*, 2, October, 2013.

[15] Ezeliora Chukwuemeka Daniel, 2Ubani Nelson O, Umeh Maryrose Ngozi, and Mbeledeogu Njide N. Time Series Decomposition Analysis of Production Quantity Using Historical data. *International Journal of Software & Hardware Research in Engineering*, 1(4), 2013, December.

[16] Ruey S. Tsay. *Analysis of Financial Time Series*, volume 712. John Wiley & Sons, 2010.

[17] Tim Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31(3):307–327, April 1986. URL http://ideas.repec.org/a/eee/econom/v31y1986i3p307-327.html.

[18] Robert F Engle. Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation. *Econometrica*, 50(4):987–1007, July 1982. URL http://ideas.repec.org/a/ecm/emetrp/v50y1982i4p987-1007.html.

[19] Eugene F. Fama. Random Walks in Stock-Market Prices. *Financial Analysts Journal*, 21:55–59, 1965.

[20] Kehinde S. James. Share Price Movement and the White-noise Hypothesis: the Algebraic Approach. *International Journal of Business & Information Technology*, 2, March 2012.

[21] Andrew W. Lo. Efficient Markets Hypothesis. In *The New Palgrave: A Dictionary of Economics*. Palgrave McMillan, New York, 2 edition, 2007.

[22] Andrew W. Lo. The Adaptive Markets Hypothesis: Market Efficiency from an Evolutionary Perspective, 2004.

[23] Dayla Mayela Pequeño Cantú. Exchange rate changes and net positions of speculators in the EuroFX futures market - Does market size matter? Master's thesis, Department of Economics Aalto University School of Economics, 2011.

[24] Eric Zivot and Jiahui Wang. *Modeling Financial Time Series with S-PLUS®*, chapter Rolling Analysis of Time Series, pages 313–360. Springer New York, 2006.

[25] Ananth Grama, George Karypis, Vipin Kumar, and Anshul Gupta. *Introduction to Parallel Computing* . Addison-Wesley, 2nd edition, January 26, 2003.

[26] Hans Moritsch. *High Performance Computing in Finance - On the Parallel Implementation of Pricing and Optimization Models.* PhD thesis, Institute of Software Technology and Interactive Systems at Vienna University of Technology, May 2006.

[27] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM*, 51(1):107–113, January 2008. ISSN 0001-0782. doi: 10.1145/1327452.1327492.

[28] Guanying Wang. *Evaluating MapReduce System Performance: A Simulation Approach.* PhD thesis, Faculty of the Virginia Polytechnic Institute and State University, August 2012.

[29] Apache Hadoop. URL http://hadoop.apache.org/.

[30] Ankit Darji and Dinesh Waghela. Parallel Power Iteration Clustering for Big Data using MapReduce in Hadoop. *International Journal of Advanced Research in Computer Science and Software Engineering*, 4:Issue 6, June 2014.

[31] Chitrank Verma. A soft introduction to Hadoop Distributed File System (HDFS) Architecture, 2014. URL http://nixustechnologies.com/2014/04/a-soft-introduction-to-hadoop-distributed-file-system-hdfs-architecture/.

[32] N. Brahmanaidu1 and Shaik Riaz . Distributed Data Storage and Retrieval on Cloud by using Hadoop. *International Journal of Science and Research*, 2012.

[33] Hadoop.apache.org. HDFS Architecure. URL http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html#Large_Data_Sets.

[34] Jiong Xie, Shu Yin, Xiaojun Ruan, Zhiyang Ding, Yun Tian, James Majors, Adam Manzanares, and Xiao Qin. Improving MapReduce Performance through Data Placement in Heterogeneous Hadoop Clusters. *In Proc. 19th Int Heterogeneity in Computing Workshop*, April 2010.

[35] Yaniv Pessach. *Distributed Storage: Concepts, Algorithms, and Implementations.* CreateSpace Independent Publishing Platform, 1 edition, February 17, 2013.

[36] Cloudera. Mountable HDFS in CDH4. URL http://www.cloudera.com/content/cloudera-content/cloudera-docs/CDH4/latest/CDH4-Installation-Guide/cdh4ig_topic_28.html.

[37] Apache Hbase. URL http://hbase.apache.org/.

[38] Nick Dimiduk and Amandeep Khurana. *HBase in Action.* Manning Publication, 1st edition, November 17, 2012.

[39] Saptarshi Guha. *Rhipe: R and Hadoop Integrated Programming Environment*, 2012. URL http://www.rhipe.org. R package version 0.73.1.

[40] S. Guha, R. Hafen, J. Rounds, J. Xia, J. Li, B. Xi, and W. S. Cleveland. Large Complex Data: Divide and Recombine (D&R) with RHIPE. *Stat*, 1(1), 2012, to appear.

[41] Divide and Recombine (D & R) with RHIPE. URL https://www.datadr.org/.

[42] Bhavani Thuraisingham. *Developing and Securing the Cloud*, chapter 12, pages 230–233. Auerbach Publications, October 28, 2013.

[43] Amazon Web Service, . URL http://aws.amazon.com/.

[44] Amazon Web Service EC2 - Previous Generation Instances, . URL http://aws.amazon.com/ec2/previous-generation/.

[45] StarCluster. URL http://star.mit.edu/cluster/.

[46] Robert E. Schapire. A Brief Introduction to Boosting. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'99, pages 1401–1406, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. URL http://dl.acm.org/citation.cfm?id=1624312.1624417.

[47] Ron Meir and Gunnar Rätsch. Advanced Lectures on Machine Learning. chapter An Introduction to Boosting and Leveraging, pages 118–183. Springer-Verlag New York, Inc., New York, NY, USA, 2003. ISBN 3-540-00529-3. URL http://dl.acm.org/citation.cfm?id=863714.863719.

[48] Michael Kearns. Thoughts on Hypothesis Boosting. December 1988.

[49] Leslie Valiant. *Probably Approximately Correct: Nature's Algorithms for Learning and Prospering in a Complex World.* Basic Books, first edition edition, June 4 2013.

[50] Leslie Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–114, November, 1984.

[51] Robert Schapire. The strength of weak learnability. *Machine Learning*, 5:197–227, 1990.

[52] H. Drucker, R.E. Schapire, and P.Y. Simard. Boosting performance in neural networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 7: 705–719, 1993.

[53] J. Friedman. Stochastic Gradient Boosting. *Computational Statistics and Data Analysis*, 38(4):367–378, 2002.

[54] Peter Bühlmann and Torsten Hothorn. Boosting algorithms: Regularization, Prediction and Model fitting. *Statistical Science*, 22:447–505, 2007.

[55] Mirko Kämpf and Jan W Kantelhardt. Hadoop.TS: Large-Scale Time-Series Processing. *International Journal of Computer Applications*, 74, 2013.

[56] Chunyang Sheng, Jun Zhao, Henry Leung, and Wei Wang. Extended Kalman Filter Based Echo State Network for Time Series Prediction using MapReduce Framework. In *Mobile Ad-hoc and Sensor Networks (MSN), Ninth IEEE International Conference on*, pages 175–180. IEEE, 2013.

[57] Leixiao Li, Zhiqiang Ma, Limin Liu, and Yuhong Fan. Hadoop-based ARIMA Algorithm and its Application in Weather Forecast. *International Journal of Database Theory & Application*, 6(5), 2013.

[58] Murray Stokely, Farzan Rohani, and Eric Tassone. Large-scale Parallel Statistical Forecasting Computations in R. In *JSM Proceedings*, 2011.

[59] H.M. Adorf. Interpolation of Irregularly Sampled Data Series–A Survey. *Astronomical Data Analysis Software and Systems IV*, 77:460–463, 1995.

[60] Andreas Eckner. A Framework for the Analysis of Unevenly-spaced Time Series Data. Technical report, Stanford University, 2012. URL http://www.eckner.com/papers/unevenly_spaced_time_series_analysis.pdf.

[61] Lutz Kilian and Mark P Taylor. Why is it so Difficult to Beat the Random Walk Forecast of Exchange Rates? *Journal of International Economics*, 60(1):85–107, 2003.

[62] Vladimir Vapnik. *Statistical Learning Theory*. Wiley, 1998. ISBN 978-0-471-03003-4.

[63] N. Sapankevych and R. Sankar. Time Series Prediction using Support Vector Machines: A Survey. *Computational Intelligence Magazine*, 4(2):24–38, 2009.

[64] Tim Hill, Marcus O'Connor, and William Remus. Neural Network Models for Time Series Forecasts. *Management science*, 42(7):1082–1092, 1996.

[65] Zhen-Guo Che, Tzu-An Chiang, and Zhen-Hua Che. Feed-forward neural networks training: a comparison between genetic algorithm and back-propagation learning algorithm. *International Journal of Innovative Computing, Information and Control*, 7(10), October 2011.

[66] R. Xiao and V. Chandrasekar. Development of a neural network based algorithm for rainfall estimation from radar observation. *IEEE Transactions on Geoscience and Remote Sensing*, 35:160–171, 1997.

[67] Liang Zhao, Sherif Sakr, Anna Liu, and Athman Bouguettaya. *Cloud Data Management*. Springer, March 25 2014.

[68] Microsoft Azure. URL https://azure.microsoft.com/en-us/.

[69] Peter Wayner. Amazon vs. google vs. windows azure: Cloud computing speed showdown, March 2014. URL http://www.computerworld.com.au/article/539633/amazon_vs_google_vs_windows_azure_cloud_computing_speed_showdown/.

[70] S. Guha, R. P. Hafen, P. Kidwell, and W. S.Cleveland. Visualization Databases for the Analysis of Large Complex Datasets. *Journal of Machine Learning Research*, 5:193–200, 2009.

[71] L. Breiman. Arcing classifiers (with discussion). *The Annals of Statistics*, 26:801–849, 1998.

[72] L. Breiman. Prediction games and arcing algorithms. *Neural Computation*, 11: 1493–1517, 1999.

[73] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting (with discussion). *The Annals of Statistics*, 28(2):337–374, 2000.

[74] P. Bühlmann. Boosting for high-dimensional linear models. *The Annals of Statistics*, 34:559–583, 2006.

[75] P. Bühlmann. Boosting with the L_2 loss: Regression and classification. *Journal of the American Statistical Association*, 98:324–339, 2003.

[76] T. Hothorn, P. Bühlmann, S. Dudoit, A. Molinaro, and M. van der Laan. Survival ensembles. *Biostatistics*, 7:355–373, 2006.

[77] L. Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.

[78] Y. Amit and D. Geman. Shape quantization and recognition with randomized trees. *Neural Computation*, 9:1545–1588, 1997.

[79] L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

[80] Y. Freund and R. Schapire. A decision-theoretic generalization of online learning and an application to boosting. In *In Proceedings of the Second European Conference on Computational Learning Theory. Lecture Notes in Computer Science, Springer.*, 1995.

[81] Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *In Proceedings of the Thirteenth International Conference on Machine Learning.* Morgan Kaufmann Publishers Inc., San Francisco, CA, 1996.

[82] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55: 119–1139, 1997.

[83] J. Friedman. Greedy function approximation: a gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232, 2001.

[84] Greg Ridgeway. Generalized boosted models: A guide to the gbm package, 2005.

[85] Greg Ridgeway with contributions from others. *gbm: Generalized Boosted Regression Models*, 2013. URL http://CRAN.R-project.org/package=gbm. R package version 2.1.

[86] Larson MG. Analysis of variance. *Circulation*, 117:115–121, 2008.

[87] R Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2014. URL http://www.R-project.org/.

[88] David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel, and Friedrich Leisch. *e1071: Misc Functions of the Department of Statistics (e1071), TU Wien*, 2014. URL http://CRAN.R-project.org/package=e1071. R package version 1.6-3.

[89] Christian Ullrich. *Forecasting and Hedging in the Foreign Exchange Markets*, volume 623 of *Lecture Notes in Economics and Mathematical Systems.* Springer, 2009.

[90] Yoav Freund and Robert E. Schapire. A Short Introduction to Boosting. *Journal of Japanese Society for Artificial Intelligence*, 14:771–780, September, 1999.

[91] Robert Schapire. Explaining AdaBoost. *Empirical Inference: Festschrift in Honor of Vladimir N. Vapnik*, 2013.

Publication

Full-length Conference Paper

Lei Li, Farzad Noorian, Duncan J.M. Moss, and Philip H.W. Leong. Rolling window time series prediction using mapreduce. In Proc. 15th IEEE International Conference on Information Reuse and Integration (IRI), pages 757–764, August 2014.