Rolling Window Time Series Prediction using MapReduce

Lei Li, Farzad Noorian, Duncan J.M. Moss, Philip H.W. Leong
School of Electrical and Information Engineering
The University of Sydney, 2006, NSW, Australia
Email: {lei.li, farzad.noorian, duncan.moss, philip.leong}@sydney.edu.au

Abstract—**Prediction of time series data is an important application in many domains. Despite their 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. We present a novel framework 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. Experimental results indicate our proposed framework scales linearly up to 32-nodes.**

Keywords. Time Series Prediction, MapReduce, Hadoop, Parallel Computing, Model Selection

I. Introduction

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]. With the increasing trend in the data size and processing algorithms' complexity, handling large-scale time series data is obstructed by a wide range of complication.

In recent years, Apache Hadoop has become the standard way to address Big Data problems [8]. The Hadoop Distributed File System (HDFS) arranges data in fixed-length files which are distributed across a number of parallel nodes. MapReduce is used to process the files on each node simultaneously and then aggregate their outputs to generate the final result. Hadoop is scalable, cost effective, flexible and fault tolerant [4].

Nevertheless, Hadoop in its original form is not suitable for time series analysis. As a result of dependencies among time series data observations [5], their partitioning and processing 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 (Fig. 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 this paper, we present a framework which addresses the above problems under the following assumptions:

- 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.
- 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.

With these assumptions, we introduce a system to allow analysis of large time series data and backtesting of complex forecasting algorithms using Hadoop. Specifically, our contributions are:

- design of a new index pool data structure for irregularly sampled massive time series data,
- an efficient parallel rolling window time series prediction engine using MapReduce, and
- a systematic approach to time series prediction which facilitates the implementation and comparison of time series prediction algorithms, while avoiding common pitfalls such as over-fitting and peeking on the future data.

The remainder of the paper is organised as follows. In Section II, Hadoop and the time series prediction problem are introduced. In Section III, the architecture of our framework is described, followed by a description of the forecasting algorithms employed in Section IV. Results are presented in Section V and conclusions in Section VI.

II. Background

A. MapReduce & Hadoop

The MapReduce programming model in its current form was proposed by Dean [6]. It centres around two functions, *Map* and *Reduce*, as illustrated in Figure 2. The first of these functions, Map, takes an input key/value pair, performs some computation 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 $(k2, list(v2)) \rightarrow list(v3)$,

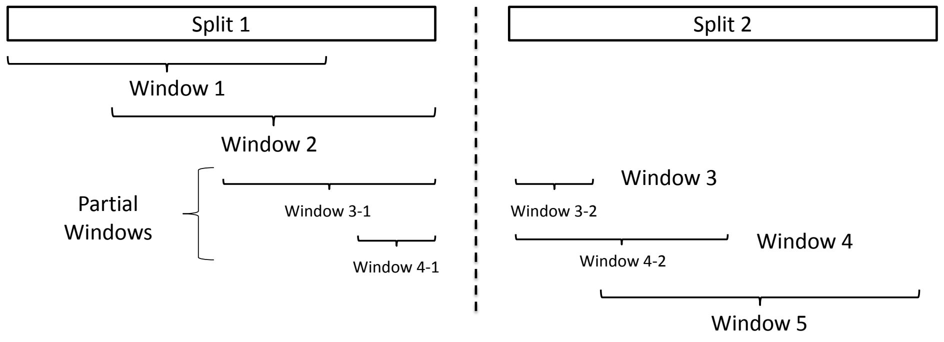

Fig. 1. Issue of partial windows: a rolling window needs data from both windows at split boundaries.

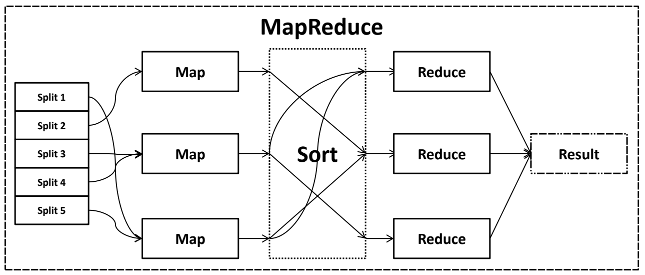

Fig. 2. Overview of parallel processing within a MapReduce environment.

takes all of the values associated with a particular key and applies some computation to produce the result. 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.

Apache Hadoop [3] is an implementation of the MapReduce framework for cluster computers constructed from computing nodes connected by a network. Used by 60% of the Fortune 500 companies, Hadoop has become the industry standard to deal with Big Data problems. The Hadoop implementation of MapReduce can be described as a cluster of TaskTracker nodes, with a JobTraker and Client node. Once a MapReduce application has been created, the job is committed to Hadoop via the Client 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 JobTrakers requirements [6]. Inputs to the map tasks are retrieved from the 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 subsequently passed to the Reduce tasks. The results of the Reduce tasks are written back to HDFS where they can be retrieved by the user [8].

HDFS file system is also the basis of Apache HBase, a column-oriented distributed database management [7]. HBase has become the common tool for big data storage and query. It originates from Google's BigTable and is developed as part of Apache Hadoop project [8]. The instinctive features of HBase are providing the capabilities of managing big data for Hadoop, such as serving database tables as the input and output for MapReduce jobs, and random real-time read/write access to data. Additionally HBase features files compression, in-memory operation and bloom filters [9].

B. Time Series & Rolling Prediction

Time series is defined as a sequence of data points observed typically at successive intervals in time [5]. Time series T can be expressed as an ordered list: $T = t_1, t_2, \ldots, t_n$ [10]. Time series data is well-suited for a number of areas such as statistical analysing, signal processing, weather forecasting, biology, mathematical economics and business management [2], [11], [12].

In time series, adjacent observation 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 [5].

Time series prediction is the use of past and current observations at time t to forecast future values at time $t+l$ [5]. Different dynamic and statistical method 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:

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

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

C. Related Work

Both the processing of times series data and specific time series prediction techniques have been previously studied by different researchers. Hadoop.TS [13] 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. [14] implemented the extended Kalman filter for time series prediction using MapReduce methodology. The framework calculates the filter's weights by performing the update step in the Map functions, whilst the Reduce function aggregates 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. [15]. 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. [16] 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 present in this paper differs from previous work in three key areas. First, we present a low-overhead dynamic model for processing irregularly sampled time series data in addition to regularly sampled data. Secondly, our framework allows applying multiple prediction methods concurrently as well as measuring their relative performance. Finally, our work 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 Hadoop framework for time series rather than the storage of massive data. The main objective of this paper is to present a fast prototyping architecture to benchmark and backtest rolling time series prediction algorithms. To the best of authors knowledge, this paper is the first paper focusing on a systematic framework for rolling window time series processing using MapReduce methodology.

III. Methodology

As in a sequential time series forecasting system, our framework preprocesses the data to optionally normalise, make periodic and/or reduce its temporal resolution. It then applies a user supplied algorithm on rolling windows of the aggregated data.

We assume that before invoking our framework, the numerical samples and their timestamps are stored on HDFS. We introduce an *index pool*, which is a table of time series indexing and time-stamp information for the entire HDFS directory. This is used to assign the appropriate index keys to time series entries, when the data could be distributed across multiple splits. 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 the 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 other partial windows 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.

Fig. 3 shows the block diagram of the architecture of the proposed system and Fig. 4 demonstrates how the data flows through the blocks. The rest of this section describes each component of the system in greater detail.

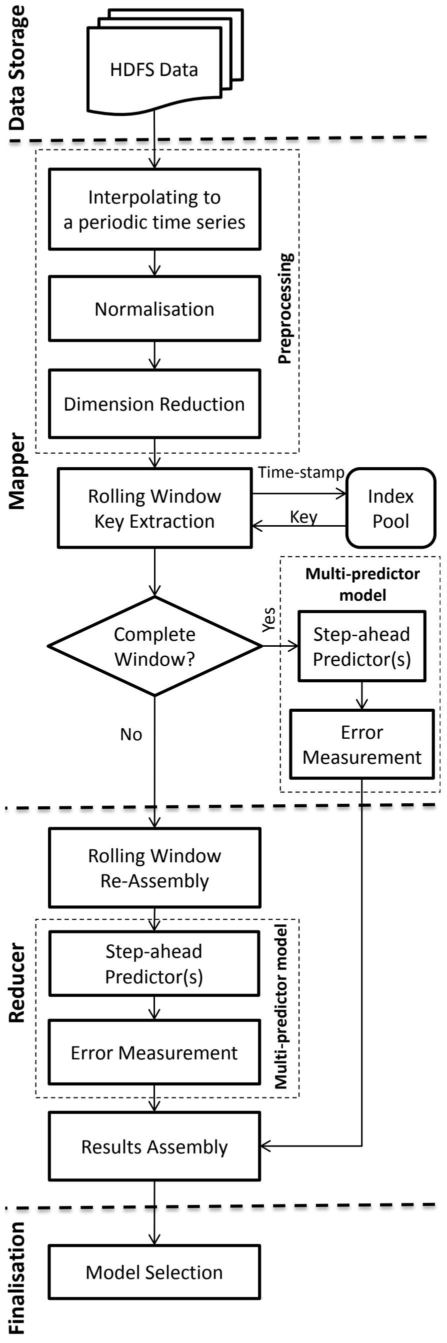

Fig. 3. The proposed system's architecture.

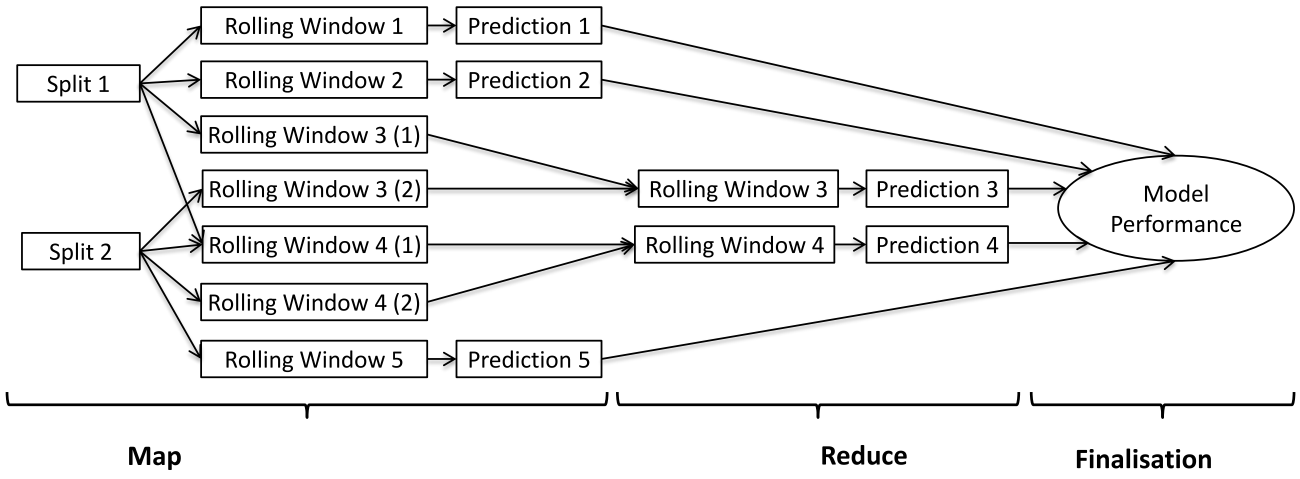

Fig. 4. Flow of data in the proposed framework.

A. Data storage and Index pool

In the proposed system, time series are stored sequentially in multiple files. Files cannot have overlapping time-stamps, and are not necessarily separated at regular intervals. Each sample in time series contains the data and its associated time-stamp. The file name is the first time-stamp of each file in ISO 8601 format. This simplifies indexing of files and data access.

Table I shows an example of an *index pool*. This is a global table that contains the start index and end index for each file as well as their associated time-stamps.

TABLE I
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 is illustrated in Fig. 3

Index pool creation is performed in a separate maintenance step prior to forecasting. Assuming that data can only be appended to the file system (as is the case for HDFS), index pool updates are fast and trivial.

B. 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 [17].
- 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 datasets include very high frequency sampling rates (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 from the aggregated time frame as used in financial Technical Analysis.

C. 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 Map 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 Fig. 4, 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.

D. 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 computation.

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, all prediction steps can be performed in the Reduce; however, this *straightforward* method is inefficient in the MapReduce methodology. 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 keys are then passed to the Reduce.

E. 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 operated on, to produce an error measure summary, allowing model comparison and selection.

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

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

$$\text{MAPE}(Y, \hat{Y}) = \frac{1}{N}\sum_{i=1}^{N}|\frac{y_i - \hat{y}_i}{y_i}| \tag{3}$$

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:

$$\text{AIC} = 2k - 2\ln(L) \tag{4}$$

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

IV. Forecasting

A. Linear Autoregressive Models

Autoregressive (AR) time series are statistical models 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].

1) AR model: A simple AR model is defined by:

$$x_t = c + \sum_{i=1}^{p}\phi_i x_{t-i} + \epsilon_t \tag{5}$$

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 $Bx_t = x_{t-1}$:

$$(1 - \sum_{i=1}^{p}\phi_i B^i)x_t = c + \epsilon_t \tag{6}$$

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, first a model is fitted to the data. Using the model, predicting the value of the next time-step is possible by using (5).

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:

$$\left(1 - \sum_{i=1}^{p}\phi_i B^i\right)(1 - B)^d x_t = c + \left(1 + \sum_{i=1}^{q}\theta_i B^i\right)\epsilon_t \tag{7}$$

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 modelled by ARIMA($0, 1, 0$) [18].

B. 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.

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:

$$\begin{aligned} s_0 &= x_0 \\ s_t &= \alpha x_{t-1} + (1 - \alpha)s_{t-1} \end{aligned} \tag{8}$$

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].

2) SVM: SVMs and their extension support vector regression (SVR) use a kernel function 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 [19]. As a result, there are several models using SVM that have been successfully studied and used in time series prediction [20].

In this paper, we use a Gaussian radial basis kernel function:

$$k(x_i, x_j) = \exp\left(\frac{-1}{\sigma^2}||x_i - x_j||^2\right) \tag{9}$$

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

We define the time series NARX model using SVM as:

$$x_t = f(C, \sigma, x_{t-1}, \cdots, x_{t-w}) \tag{10}$$

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.

We use the following algorithm to perform cross-validation:

```
for w in $[w_{min}, \dots, w_{max}]$ do
    prepare $X$ matrix with lag $w$
    training_set ← first 80% of $X$ matrix
    testing_set ← last 20% of $X$ matrix
    for $C$ in $[C_{min}, \dots, C_{max}]$ do
        for $\sigma$ in $[\sigma_{min}, \dots, \sigma_{max}]$ 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) 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 [21]. 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. A training algorithm such as Back-propagation is used to minimise error of this network's output. Similarly a cross-validation algorithm is used for selecting this hyper-parameter.

In this paper, a feed-forward ANNs with a single hidden layer is used.

V. Results

A. Experiment Setup

This section presents an implementation of the proposed framework in R programming language [22]. R was chosen as it allows fast prototyping, is distributed under GPL license and includes a variety of statistical and machine learning tools in addition to more than 5600 third party libraries available through CRAN [23].

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 [24]. The statistical computing software R version 3.1.0 [22] and RHIPE version 0.73.1 [25] were installed on each Ubuntu image.

2) RHIPE: R and Hadoop Integrated Programming Environment (RHIPE) [25] introduces an R interface for Hadoop, in order to allow analysts to handle big data analysis in an interactive environment. We used RHIPE to control the entire MapReduce procedure in the proposed framework.

3) Test Data: In order to test the framework, synthetic data was generated from an autoregressive (AR) model using (5). The order $p = 5$ was chosen and $\{\phi_i\}$ were generated randomly. The synthetic time series data starts from 2004-01-01 to 2013-12-31, in 30 minute steps, and is 6.6 MB in size.

4) Preprocessing and window size: The preprocessing included normalisation, where the samples were adjusted to have average of $\mu = 0$ and standard deviation $\sigma = 1$. All tests 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 the prediction procedure in the framework. Consequently, each predictor model was trained on the previous 3 hours of data to forecast 30 minutes ahead.

B. Test Cases

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

Table II shows the execution time for this test. With consideration for the capacity of *m1.large*, we limited the maximum number of Map and Reduce tasks in each node to the number of CPUs and half the number of CPUs respectively. As evident in Table II, scaling is approximately linear up to 32 nodes for the reasonably small example tested. In smaller clusters, the proposed framework is considerably well-scaled. Furthermore, the speed-up of 16 and 32 nodes,are 13.09 and 30.83 respectively. While these speed-ups are not perfect, they are in an acceptable range. We expect a similarly good scaling for larger problems.

TABLE II
AWS EC2 EXECUTION TIMES SCALING.

Cluster Size	Mappers	Reducers	Total (Sec)	Speed-up
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

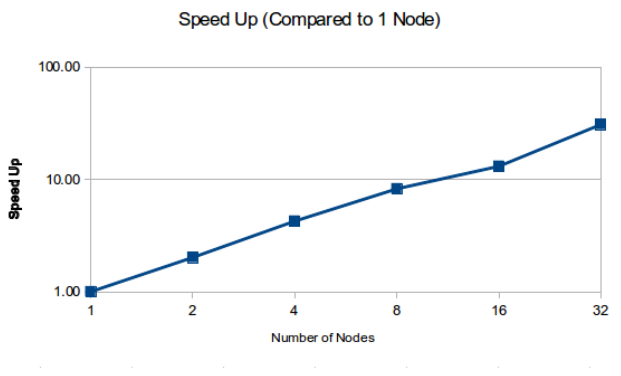

Fig. 5. Speed-up of execution time versus cluster size.

2) 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 of 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 [26] for ARIMA, ANN, Naïve and ETS models, and e1071 package [27] 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 IV-B.

Table III 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. 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$\times$ faster than executing all models individually,

TABLE III
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

as the former requires significantly less disk 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.

3) 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 III-D, was implemented without this algorithm.

TABLE IV
COMPUTATIONAL EFFICIENCY OF THE PROPOSED ARCHITECTURE.

Cluster size	Proposed framework (s)	Benchmark framework (s)	Improvement %
16 Nodes	4351.05	5359.92	%123
32 Nodes	2444.17	3566.94	%145

Table IV 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. The significant advantages in execution time can be explained due to the reduced overhead of data being cached before being moved from the Map to the Reduce, therefore reducing the communication time.

VI. CONCLUSION

We presented an efficient framework for rolling window time series prediction, with a focus on computational efficiency for large-scale data. By utilising an index pool, windows occurring across data splits can be easily reassembled via the sorting stage of MapReduce. Our approach allows users to only be concerned about designing prediction algorithms, with data management and parallelisation being handled by the framework.

VII. Future Work

We are planing to apply our framework with other time series analysis methods like classification and data mining. Furthermore, we are trying to implement the proposed system for a comparatively larger time series data set and further optimize it for both better performance and functionality.

Acknowledgement

This research was partially supported by Australian Research Council's Linkage Projects funding scheme (project number LP110200413) and Westpac Banking Corporation. We also thank Dr. Mahsa Baktashmotlagh for her valuable comments.

References

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

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

[3] Apache Hadoop. [Online]. Available: http://hadoop.apache.org/

[4] IBM Software. What is Hadoop? [Online]. Available: www.ibm.com/software/data/infosphere/hadoop/

[5] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel, *Time Series Analysis: Forecasting and Control*, 4th ed. Wiley Series in Probability and Statistics, June 30, 2008.

[6] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[7] Apache Hbase. [Online]. Available: http://hbase.apache.org/

[8] R. 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.

[9] N. Dimiduk and A. Khurana, *HBase in Action*, 1st ed. Manning Publication, November 17, 2012.

[10] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. 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*, 2012, pp. 262–270.

[11] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*. Otexts, 2013. [Online]. Available: https://www.otexts.org/fpp

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

[13] M. Kämpf and J. W. Kantelhardt, "Hadoop.TS: Large-Scale Time-Series Processing," *International Journal of Computer Applications*, vol. 74, 2013.

[14] C. Sheng, J. Zhao, H. Leung, and W. 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*. IEEE, 2013, pp. 175–180.

[15] L. Li, Z. Ma, L. Liu, and Y. Fan, "Hadoop-based ARIMA Algorithm and its Application in Weather Forecast," *International Journal of Database Theory & Application*, vol. 6, no. 5, 2013.

[16] M. Stokely, F. Rohani, and E. Tassone, "Large-scale Parallel Statistical Forecasting Computations in R," in *JSM Proceedings*, 2011.

[17] H. Adorf, "Interpolation of Irregularly Sampled Data Series–A Survey," *Astronomical Data Analysis Software and Systems IV*, vol. 77, pp. 460–463, 1995.

[18] L. Kilian and M. P. Taylor, "Why is it so Difficult to Beat the Random Walk Forecast of Exchange Rates?" *Journal of International Economics*, vol. 60, no. 1, pp. 85–107, 2003.

[19] V. Vapnik, *Statistical Learning Theory*. Wiley, 1998.

[20] N. Sapankevych and R. Sankar, "Time Series Prediction using Support Vector Machines: A Survey," *Computational Intelligence Magazine*, vol. 4, no. 2, pp. 24–38, 2009.

[21] T. Hill, M. O'Connor, and W. Remus, "Neural Network Models for Time Series Forecasts," *Management science*, vol. 42, no. 7, pp. 1082–1092, 1996.

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

[23] The Comprehensive R Archive Network. [Online]. Available: http://cran.r-project.org/

[24] Amazon Web Service EC2. [Online]. Available: http://aws.amazon.com/ec2/previous-generation/

[25] S. Guha, *Rhipe: R and Hadoop Integrated Programming Environment*, 2012, R package version 0.73.1. [Online]. Available: http://www.rhipe.org

[26] R. J. H. with contributions from George Athanasopoulos, S. Razbash, D. Schmidt, Z. Zhou, Y. Khan, C. Bergmeir, and E. Wang, *forecast: Forecasting Functions for Time Series and Linear Models*, 2014, R package version 5.3. [Online]. Available: http://CRAN.R-project.org/package=forecast

[27] D. Meyer, E. Dimitriadou, K. Hornik, A. Weingessel, and F. Leisch, *e1071: Misc Functions of the Department of Statistics (e1071), TU Wien*, 2014, R package version 1.6-3. [Online]. Available: http://CRAN.R-project.org/package=e1071