A distributed real-time data prediction framework for large-scale time-series data using stream processing

Kehe Wu, Yayun Zhu, Quan Li and Ziwei Wu
School of Control and Computer Engineering, North China Electric Power University, Beijing, China

Received 13 September 2016
Revised 22 November 2016
13 January 2017
Accepted 5 March 2017

Abstract

Purpose – The purpose of this paper is to propose a data prediction framework for scenarios which require forecasting demand for large-scale data sources, e.g., sensor networks, securities exchange, electric power secondary system, etc. Concretely, the proposed framework should handle several difficult requirements including the management of gigantic data sources, the need for a fast self-adaptive algorithm, the relatively accurate prediction of multiple time series, and the real-time demand.

Design/methodology/approach – First, the autoregressive integrated moving average-based prediction algorithm is introduced. Second, the processing framework is designed, which includes a time-series data storage model based on the HBase, and a real-time distributed prediction platform based on Storm. Then, the work principle of this platform is described. Finally, a proof-of-concept testbed is illustrated to verify the proposed framework.

Findings – Several tests based on Power Grid monitoring data are provided for the proposed framework. The experimental results indicate that prediction data are basically consistent with actual data, processing efficiency is relatively high, and resources consumption is reasonable.

Originality/value – This paper provides a distributed real-time data prediction framework for large-scale time-series data, which can exactly achieve the requirement of the effective management, prediction efficiency, accuracy, and high concurrency for massive data sources.

Keywords Prediction, Real-time, Autoregressive integrated moving average, Storm, Stream processing, Time series

Paper type Research paper

1. Introduction

Recently, with significant advancements in smart meters/sensors and communication technologies, sensors networks have been widely used in many domains, such as environmental monitoring (Trilles *et al.*, 2015), traffic analysis (Liu *et al.*, 2006; Zhao *et al.*, 2015), and electric equipment maintenance (Fonseca *et al.*, 2008; Liu *et al.*, 2012). All these applications require continuous monitoring and periodic acquisition data produced by sensors, and therefore will generate variety of time-series data. Time-series data from complex systems capture the dynamic behaviors and causalities of the underlying processes and provide a tractable means to predict and monitor system state evolution (Cheng *et al.*, 2015). Under this circumstance, analyzing and mining of time-series data becomes a critical issue.

Prediction, as an important aspect of data mining, may help a lot to detect and alleviate system anomalies by correlating new incoming data with historical observations. Prediction has been investigated within some fields for time-series data, and benefits are obtained due to accurate prediction: a new on-condition maintenance module applied in wind generators to optimize the cycles of production (Fonseca *et al.*, 2008); gaming workload prediction for good game quality and power savings (Dietrich *et al.*, 2015);

This paper is supported by "the Fundamental Research Funds for the Central Universities (2015XS72)."

International Journal of Intelligent Computing and Cybernetics
Vol. 10 No. 2, 2017
pp. 145-165

1756-378X
DOI 10.1108/IJICC-09-2016-0033

data from World Wide Web being used to predict nonfarm payrolls, which is considered as an important indicator of economy (Levenberg *et al.*, 2013); and prediction of the coming traffic of highway to raise abnormal events alarms and avoid traffic jams (Liu *et al.*, 2006). However, time-series data prediction is still a major research direction (Fu, 2011; Kejariwal *et al.*, 2006); actually, forecasting the evolution of complex systems is noted as one of the ten grand challenges of modern science (Cheng *et al.*, 2015).

A big challenge faced by precise prediction is real-time requirement. In an electric power system, smart meters send data in a 15-minute interval to the data center (Bose, 2010), and phasor measurement units can help the operators to measure the values of voltage and current accurately in very short time intervals (typically 30-60 phasors per second) (McHann, 2013). Similarly for environment monitoring, equipment abnormal warning, and other applications, elements of data need to be processed in real time; else one may lose the opportunity to process them at all (Kejariwal *et al.*, 2006).

Another challenge is the operation of large-scale data sources, because many applications are rather complex and consist of multiple data streams. In mission operations for NASA's Space Shuttle, approximately 20,000 sensors are telemetered once per second to Mission Control at Johnson Space Center, Houston (Keogh and Smyth, 1997); there are about 50,000 securities trading in the USA, and every second up to 100,000 quotes and trades (ticks) are generated (Zhu and Shasha, 2002). In 2011, an estimated 493 utilities in the USA had collectively installed more than 37 million smart meters (Chen *et al.*, 2015); just three years later, it went up to almost 58.5 million (EIA, US, 2014).

Unfortunately, state-of-the-art technologies are difficult to handle these challenges since several characteristics are raised:

(1) the number of events in time series can be gigantic, making it difficult to manage and query data (Cui *et al.*, 2015);

(2) many time-series queries are themselves complex, and involve intricate statistical analysis such as correlation, e.g., active power and electric current in Power Grid data; and

(3) time-series data are periodically arrived and real-time response is required; the previous methods incur a very high update cost for either mining fuzzy rules or training parameters in different models, which may not work in the stream scenarios.

In this paper, a distributed real-time data prediction framework for large-scale time-series data is proposed. This framework employs a stream processing technique and can exactly achieve the requirement of the prediction efficiency, accuracy, and high concurrency for massive data sources. Specifically, Power Grid monitoring data is typical time-series data, the number of monitored sensors is stupendous, and its real-time requirement is relatively high, as stated above. For this reason, this paper takes the Power Grid data as an example to establish the framework and conduct experiments. Because the autoregressive integrated moving average (ARIMA) prediction algorithm has the advantages of broad applicability, short training time consuming, relatively high prediction accuracy, etc., which can support the basic properties of Power Grid data and fulfill the requests of various types of data modeling, it is adopted in this paper.

The contributions of the paper are as follows:

(1) several prediction algorithms are compared and a fast self-adaptive algorithm is adopted to fit a large number of data sources;

(2) an effective time-series data storage system is built based on HBase, a database implemented on top of the Hadoop stack;

(3) a real-time and parallel prediction framework is realized by Storm, which is one of the most popular stream processing systems; and

(4) a proof-of-concept testbed is illustrated to verify the proposed framework.

The remainder of this paper is organized as follows: Section 2 presents an overview of related works about time-series prediction; a self-adaptive algorithm is adopted in Section 3 to fit a large number of data sources; Section 4 contains our real-time and parallel calculation framework; Section 5 illustrates several experiments; and then discussion and conclusion are presented in Section 6.

2. Related works

2.1 Prediction algorithms

Predicting unknown values of time series is defined as follows: given a series of consecutive values $\{X_t\} = x_1, x_2, \ldots, x_t$, and the next value x_{t+1}, even x_{t+2}, x_{t+3}, … can be predicted through some specific algorithms. A lot of work on time-series prediction has been conducted to date. Related works can be divided into data mining approaches (Cheng *et al.*, 2015; Dietrich *et al.*, 2015; Fonseca *et al.*, 2008; Velasquez-Henao *et al.*, 2012), fuzzy rules (Policker and Geva, 2000), and similarity methods (Lian and Chen, 2008; Shibuya *et al.*, 2009).

For data mining techniques, Wagner *et al.* (2011) developed a system for a large number of time series to forecast on the order of 10^5, which made use of a novel hybrid model based on the AR(7) model for time-series modeling, analysis, and prediction; practical applications had proved it to be robust. However, the system was designed without consideration of the real-time requirement for abundant applications. Fonseca *et al.* (2008) proposed an on-condition maintenance approach for wind generators, in which the prediction models used regression techniques based on SVR, ARMA, and ARIMA. Dietrich *et al.* (2015) searched time-series models for workload prediction for gaming applications and other remarkable works, prediction algorithms like PID, LMS, and ARMA, all revealed with very encouraging results. Cheng *et al.* (2015) reviewed the recent developments in nonlinear and non-stationary time-series forecasting and provided a comparative evaluation of the performance of alternative models in real-world application case studies, especially where traditional models (ARMA and KF) failed to capture the system evolutions. Velasquez-Henao *et al.* (2012) proposed a modification of the dynamic architecture for artificial neural networks (ANNs) (DAN2) model and proved it outperformed the original DAN2 model, which performed significantly better than traditional ANNs.

The traditional fuzzy prediction algorithm cannot guarantee the prediction accuracy if the time-series' statistics are non-stationary. As an improvement, Policker and Geva (2000) used the existing fuzzy clustering algorithm, the unsupervised optimal fuzzy clustering algorithm, and the deterministic annealing approach to classify a large set of time series such that the series in each cluster were close and their temporal probabilistic behavior was similar. As a second step, the proposed algorithm predicted the future data from a mixture probability distribution function. In particular, for each future value, the fuzzy membership of each cluster was calculated, and the results were combined to produce an estimate of this future value.

For similarity methods, Shibuya *et al.* (2009) proposed a new method for quantifying the strength of the causal influence from one time series to another, whether each of two time series was symbolic or numerical. The proposed method could be used to predict one series depending on the strength of causality with another series. Lian and Chen (2008) proposed three approaches, polynomial, discrete Fourier transform, and probabilistic, to predict the unknown values that had not arrived at the system and answered similarity queries based on the predicted data.

Table I contrasts different prediction algorithms in four aspects including training difficulty, applicability, operation efficiency, and accuracy. As we can see, the ARIMA model has the advantages of broad applicability, short training time consuming, relative high prediction accuracy, and so on; thus it is adopted as the prediction algorithm in our proposed framework.

2.2 Stream processing frameworks

2.2.1 Storm. Storm (ASF, 2016d) is a distributed real-time computing framework that can handle unbounded data stream easily and reliably until it is artificially stopped. As a high-reliability flow computing framework, Storm is an open source by Twitter.

Storm has its own terminology, starting with the concept of Topologies. A Topology defines the way, or workflow, in which a problem will be processed. It is similar to a job in Hadoop (ASF, 2016a). However, Hadoop jobs will have an end, while the Topology will always run because there is a need for continuous computation. Figure 1 illustrates a Topology as the workflow of Spouts and Bolts. The Spouts handle the insertion of data tuples into the Topology and send each of these tuples to Bolts. They thus connect the input stream and send the values to the Bolt that is responsible for the stream processing. Each Bolt processes the streams that it receives from the Spout. It applies any particular procedure to generate its output stream. The actual improvement of Storm compared to other solutions is that these operations can be parallelized. Parallelization is handled on the level of a single Bolt and the parallelization factor is defined as part of the Topology.

2.2.2 Spark Streaming. Spark Streaming (ASF, 2016c) is another famous stream processing framework, which allows scalable, high-throughput, fault-tolerant stream processing of live data streams. Data can be ingested from many sources like Kafka, Flume, Twitter, Kinesis, or plain TCP sockets and be processed using complex algorithms expressed with high-level functions like map, reduce, join, and window. Finally, processed data can be pushed out to file systems, databases, and live dashboards. Spark Streaming provides a high-level abstraction, called discretized stream or DStream, which represents a

Table I.
Contrast of common prediction algorithms

Algorithms	Training difficulty	Applicability	Operation efficiency	Accuracy
ARIMA	Easy	Wide	High	Medium
ANN	Medium	Wide	Medium	Medium
DAN2	Medium	Wide	Medium	High
Fuzzy algorithms	Easy	Medium	Medium	Medium
Similarity methods	Hard	Medium	Low	Medium

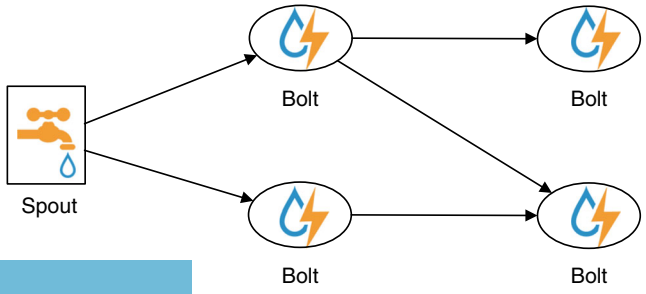


Figure 1.
An example of Storm, its Topology includes one Spout and four Bolts

continuous stream of data. DStreams can be created either from input data stream from sources such as Kafka, Flume, and Twitter, or by applying high-level operations on other DStreams. More details about Spark Streaming are presented by Drusinsky (2016).

3. Data prediction model

Since large-scale time-series data prediction requires frequent model training over time, a general model which can update itself, capture, and predict a wide variety of time series without human intervention is necessary. Additionally, the computation time to generate forecasts should be as short as possible; in other words, efficiency is a significant issue. Among the existing prediction algorithms, as shown in Table I, the ARIMA model has the advantages of broad applicability, short training time consuming, and relatively high prediction accuracy. Thus, we use ARIMA as a prediction algorithm in this work.

3.1 Brief introduction of ARIMA

ARIMA is a widely used method on real-time predication. A time series $\{X_t\}$ which satisfies ARIMA (p, d, q) is shown in the following equation:

$$X_t = \frac{\theta(B)\varepsilon_t}{\varphi(B)\nabla^d}. \tag{1}$$

In Equation (1), the symbol ∇ refers to the difference operator and $\nabla^d = (1-B)^d$, while d is the difference order. B indicates the delay operator and satisfies $BX_t = X_{t-1}$, $\varphi(B)$ and $\theta(B)$ express the reversible operator and stationary operator, and they are satisfied, respectively, with the following two equations:

$$\varphi(B) = 1-\varphi_1 B-\varphi_2 B^2-\cdots-\varphi_p B^p. \tag{2}$$

$$\theta(B) = 1-\theta_1 B-\theta_2 B^2-\cdots-\theta_q B^q. \tag{3}$$

In Equations (2) and (3), p and q are models' orders, and φ and θ are the corresponding parameters of the models. ε_t is zero mean white noise sequences. The following equation is adopted to predict the value of time series at the next moment:

$$X_{t+1} = \frac{\theta(B)\varepsilon_{t+1}}{\varphi(B)\nabla^d}. \tag{4}$$

In a time-series analysis, the most crucial step is to identify an appropriate structure and estimate the parameters of the ARIMA model for a given time-series data. Model identification can be divided into three steps (Box *et al.*, 2013). The first step is to analyze the observed time-series data and determine the required type of transformation (differencing and/or power transformation, if necessary). The second step is to identify the corresponding ARIMA model structure by comparing the estimated autocorrelation coefficient (ACC) and partial autocorrelation coefficient (PACC) of the transformed data with the theoretical ACC and PACC. The third and last step is to determine the deterministic trend term such that the mean of the simulated time-series data is equal to the mean of the observed time-series data.

Once the model structure is identified, the model parameters can be estimated by using the maximum likelihood estimator. A diagnostic check is then made on the resulting model. This can be done by using the ACC and PACC of the residual where both have to be within the critical limits for time lags different from 0. We refer the readers to Box *et al.* (2013) for more details about the parameters' determination of the ARIMA model.

3.2 Pre-build model of time-series data

Although the time-series data model of any single data source is not immutable, however, within a specific time window, data features keep steady. Taking the data monitored by various sensors in Power Grid as an example, we can obtain three kinds of different rules:

(1) Fitted by ARIMA (p, q), amplitude variation of data is small; usually this series is a stationary sequence, such as tension of wire and C_2H_2 in oil.

(2) Fitted by ARIMA (p, d, q), data rising or falling slowly, which can be transformed into a smooth sequence and then fitted by ARIMA (p, q), for example, CO and CO_2 in oil, currents of wire, etc.

(3) Fitted by ARIMA (p, d^s, q), data is changed periodically; similar data reappear after interval s, for instance, temperature of oil, temperature of wire, active power, and so on.

Based on this premise, we can use historical data to estimate the parameters of each time series, and preserve in advance. This set of parameters can be used repeatedly for data prediction at a fixed time interval. As a result, the training speed is improved and the prediction time is reduced.

4. Processing framework

4.1 Overall design of the framework

Time-series data are usually continuous production at a fixed interval, e.g., Power Grid data center collected data from smart meters in every 15 minutes (Bose, 2010). It will significantly benefit the actual analysis if prediction data of time t, labeled as P_t, is ready when actual measurement data of time t, M_t, arrives. This inspires the original design of our framework, which is shown in Figure 2.

Suppose we use last N values (V_{t-N}, V_{t-N+1}, …, V_{t-1}) in time series to predict the next value. At $t-1$ moment, V_{t-N}, V_{t-N+1}, …, V_{t-1} are ready and fed into the ARIMA model to forecast P_t. Then, at t moment, after M_t arrives, P_t and M_t are compared to detect abnormal data, and the data governance method is used to determine a value V_t as the accepted value at t moment. The above process is an iterative procedure as time goes by. So Storm is introduced into our framework to undertake this mission, due to its reliably performance on continuous computation for unbounded data streams.

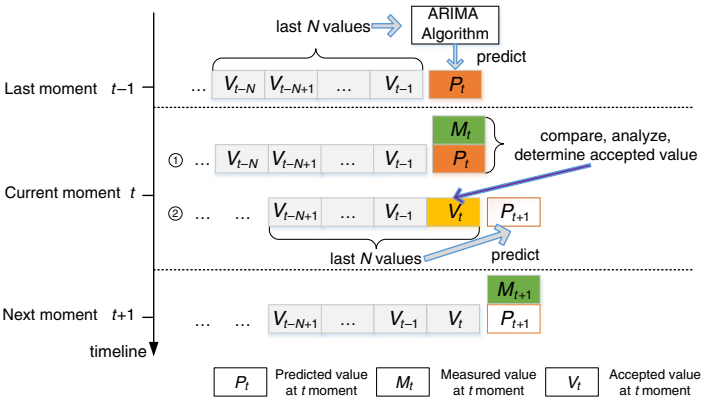


Figure 2.
Overall design of this framework

Note: This process is only for a single data sources

4.2 Storm-based platform

In our framework, the task of Storm is reading massive monitoring data, coordinating the models' training of different time series, real-time analysis of the result of prediction, storing results, and so on. Figure 3 shows Topology of this framework, which is made up of distributed concurrent execution of multiple Spouts and many sets of Bolt, and each Bolt group contains pretreatment, ARIMA prediction, etc. The tuples launched by Spout are key-value pairs which are composed of data source type and several time-series data in recent moments. Here, we choose Group by Field as a grouping strategy, to render each Bolt group only receive certain specific data sources for processing. For example, a group of Bolt is only in charge of voltage sensors as data sources and ignore other data. The significance of adopted grouping strategy is to divide massive and various variety of data sources into multiple sets of Bolts, using concurrent processing to speed up prediction velocity and raise convenience of managing considerable data sources.

4.3 Time-series data storage model based on the HBase

The traditional time-series data storage model is relatively simple, which contains three columns, as represented in Table II. In the traditional model, each record stands for the measured value/state of one measure unit (sensor) in one moment.

Since the number of measure units can be stupendous, and measured interval is bitty, vast data needs to be written into the database continuously. This is an extremely heavy burden, and might influence the performance of data reading. Besides, the time measure units gather data that is synchronous in practice. In this case, the traditional model always stores a series of timestamps, which occupy mickle storage space.

Our framework uses HBase to simulate data sources. HBase (ASF, 2016b) is an open source project of Apache, using HDFS as its file system. It is very suitable for the sequential data because of the flexible, loose storage structure.

To facilitate the management of massive amounts of data, we design the data storage structure based on HBase shown in Figure 4, which contains an index table and some data tables.

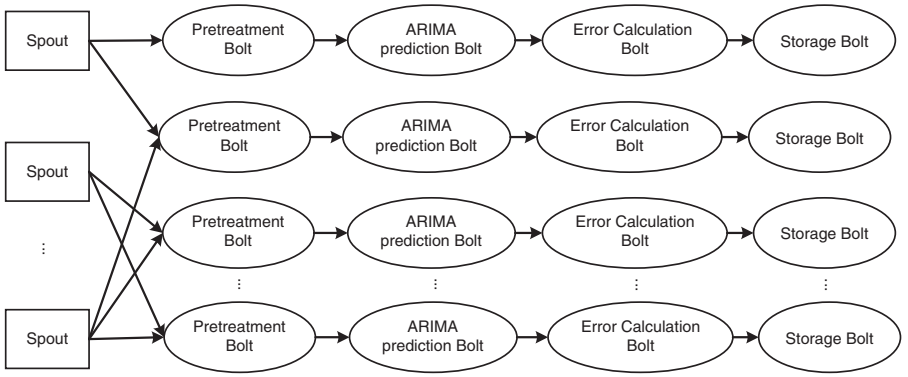


Figure 3.
Topology of storm in this framework

Table II.
Traditional storage model for time-series data

Timestamp	Name of measure unit	Value/state of measure unit

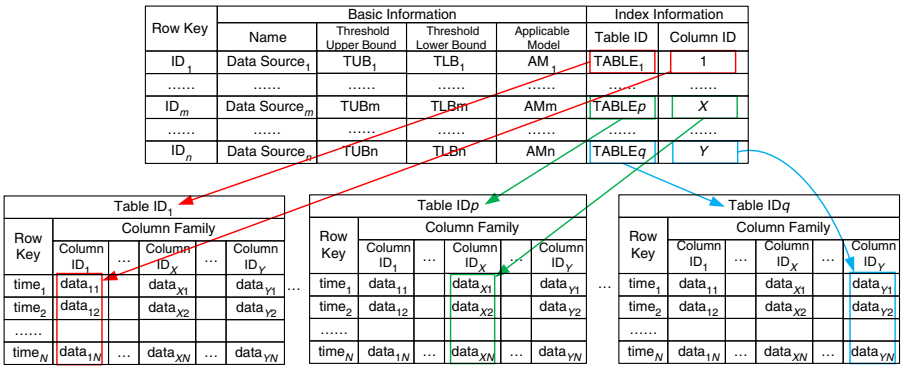


Figure 4.
Time-series data storage structure based on HBase

Basic information of data sources and index information are recorded in the index table, including name of the monitoring sources (name), the upper and lower bounds of the error warning threshold value (Threshold Upper and Lower Bound), applicable fitting model (applicable ARIMA model) and so on, and the index information, including the table number (Table ID) of storing the monitoring data and the column number (Column ID) in the table.

Each column of data table represents the time-series data which is collected by one data source. Its table number and column number are corresponding to the index information (Table ID and Column ID) in the index table. The key of data table is the timestamp when time-series data is gathered, to simulate the time series by means of line growth. Actually, real-time data of each data source will be written into HBase at a fixed interval.

Compared to the traditional storage model, only one record is needed for all measured values at the same moment in the proposed model, which decreases the load and increases the use efficiency of the database. Due to the specific feature of HBase, multi-versions of data are stored in the same cell, which facilitates the management of data quality since it needs original data and fixed data both.

4.4 Real-time framework for distributed time-series prediction

In order to meet the demand of real-time and large-scale time-series prediction, we design a distributed framework based on Storm, using HBase as a data source. Figure 5 shows this diagram. Spout and Bolt in Storm are responsible for data collection and processing, respectively. Due to the huge number of data sources, we use multiple Spouts to read data parallel and ensure that each Spout reads the same number of data sources, avoiding the emergence of the hot spots, which means several Spouts are much busier than others, and thus might lead to a bottleneck. According to Topology as shown in Figure 3, each Bolt group only selects a specific type of data sources from all Spouts to receive, process, and analyze. The mechanism of Spout and Bolt will be demonstrated in detail below.

4.4.1 The mechanism of Spout. Spout is an active role, which communicates with HBase by calling the method *nextTuple()* to read data constantly, make up different data sequences according to data sources, and then send them to Bolt for later processing. The procedure of Spout is shown in Figure 6:

- Step 1: initialize the running environment, connect HBase, and iterate over the collection of data sources which are collected by current Spout. Then look up Table ID and Column ID corresponding to each data source according to the index table of HBase. After that read N recent data (as shown in Figures 2 and 4) from the data table by Table ID and Column ID, and build the initial time series in the form of a linked list

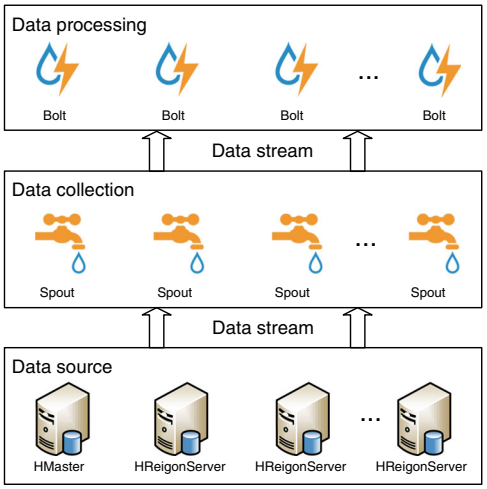


Figure 5.
Diagram of prediction framework

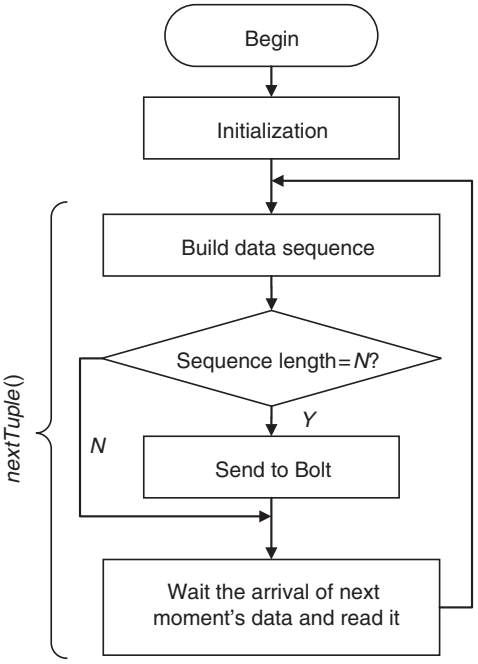


Figure 6.
Procedure of Spout

noted as Link, where N means the sample size of historical data to predict the next value (same with N in Section 4.1). In order to speed up reading efficiency, the framework will cache Table ID and Column ID in the corresponding Spout's memory to reduce index time;

- Step 2: build the serial Link; if its length reaches N, then turn to step 3, otherwise skip to step 4;

- Step 3: launch Link in the form of a tuple to the Bolt according to the grouping strategy; and
- Step 4: read new data from HBase after the arrival of next moment's data, and then go to step 2.

Steps 2-4 constitute the method *nextTuple()* of Spout. In the process, two kinds of methods to build a data sequence are shown as follows (corresponding to Figures 7(a) and (b)):

(1) in initialization, if the length of data sequence is not up to *N*, use the tail-plug method to add new data to the end of the list; and

(2) within running processing, when the length of the data sequence equals *N*, use the tail-plug method to add new data at the end of the list, and delete the first element of the list at the same time.

4.4.2 The mechanism of Bolt. This framework uses multiple sets of Bolts which receive tuple from Spout. Each group of Bolts includes operation such as preprocessing, prediction, error calculation, etc. Bolt is a passive role which calls the *execute()* function after receiving message, and handles received tuples according to the established grouping strategy.

Each group of Bolts first executes pretreatment operation including tuple splitting, formatting by the requirement of prediction algorithm, and so on. Second, the ARIMA algorithm in Section 3.1 is embedded in *execute()*. After Bolt receives the data sequence from the pretreatment stage, ARIMA is called automatically to estimate the data of the next moment, thus predicting the time sequence in real time. Then Bolt trains the model according to the type prestored in the index table. When the default model type does not meet the characteristics of this sequence, Bolt also provides a model switch function until it gets a best-fitted one, which could effectively avoid blindness and enhance the efficiency of prediction. It is necessary to store forecasting data temporarily before real measurement data of the next moment being collected. Then Bolt sends two values together to the error calculation Bolt and analyze the results. Third, the error analysis Bolt calculates the forecast error according to the actual data and forecast data which are sent by prediction Bolt, and refers the error threshold been recorded in index table to decide whether to warn. Finally, storage Bolt stores results of prediction, error information, and warning information, which would be convenient to further analyze. Process flow of Bolt group is shown in below.

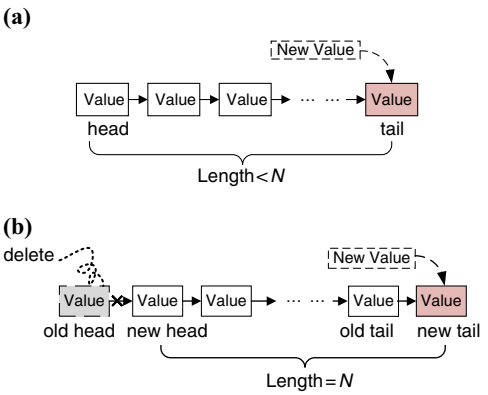


Figure 7.
The method of building data sequence in two cases

Notes: (a) Initialization method of building data sequence; (b) the method of building data sequence while running

```
Pretreatment bolt
   Receive_tuples();//receive the tuples sent by Spout
   Process_tuples();//process tuples to adapt ARIMA algorithm
ARIMA prediction bolt
   Train_model();//train model as pretreated
   While (!pass_diagnosis) {//check the model whether is suitable or not
      Adjust_model();//adjust model
      Train_model();//train model
   }//until the model is suitable
   Predict();//predict the data of next moment
Error calculation bolt
   Error_calculation();//according the prediction data of last moment and the actual data
   of current moment to calculate error
   if(error > threshold) //error is over threshold
      alert();// warning
Storage bolt
   Save_results();//store the result of prediction, predict error, warning information
   and others
```

5. Experiments

In order to verify the effectiveness of the proposed framework in practical application, we deploy a proof-of-concept cluster which contains nine nodes and each node is based on the low-cost credit-card-sized single-board PC (Raspberry PI, model 2B, hardware configuration in Table III) (RaspberryPIFoundation, 2015), as shown in Figure 8. Since Storm is architected as a Master-Slave structured system, in our proof-of-concept cluster, we choose one to be regarded as Master and the rest as Workers. Considering the reusability of the cluster for HBase, we place HMaster (in HBase) to the same node with Master, and HRegions (in HBase) to the same nodes with Workers.

The experiment selects 500 monitoring sensors from a city Power Grid as actual data sources; most of them are measured for Electric Current, Active Power and Reactive Power

Type	Model/Parameter
CPU	ARM Cortex-A7 (900MHz, 4cores)
RAM	1 GB
Hard-disk	64 GB
Operation system	Linux (CentOS 7 for ARM)
Network	100 M bps

Table III. Hardware configuration

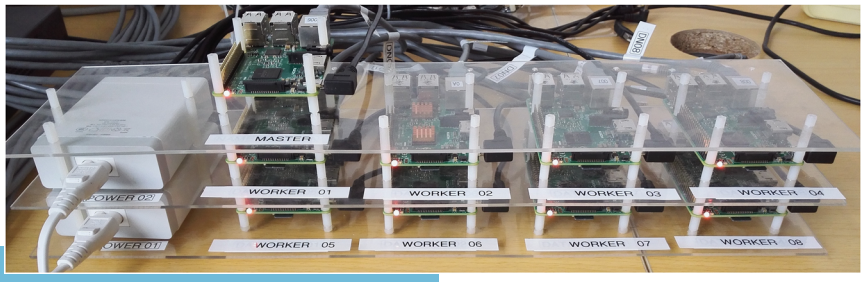


Figure 8. A proof-of-concept demonstration

in a transformer. Measured data is collected in every 5 minutes. We divide all data sources into ten groups, and each group contains 50 sources. Also, we set ten Spouts and ten Blots in Storm.

Our experimental cluster runs two days' continuous measured data, so the amount of value to be predicted for each measurement sensor is: $(2 \times 24 \times 60)/5 = 576$. Experimental evaluation includes six parts, which are the influence of sample size on the prediction result, the accuracy analysis, the processing efficiency of Spout and Bolt in Storm, the resource consumption of servers, effect of the number of nodes and performance comparison between different algorithms.

5.1 The influence of sample size on the prediction result

Section 4.4.1 points out that each Spout forms a time sequence by reading the N latest values and launches it to Bolt for prediction. Since the ARIMA algorithm uses historical data to predict, the bigger the sample size (N in this paper), the better is prediction result. However, it is not realistic in practical applications. This section will focus on the impact of sample size, i.e., N, on the prediction result, in order to select the best N for subsequent experiments. In total, 20 monitoring sources are randomly selected from 500 and divided equally into five groups, and then the mean relative error (MRE) is calculated (see Equation 6); results are shown in Figure 9.

It can be concluded from Figure 9 that the prediction error of each group is gradually reducing while N increases, and when N goes up to 40, errors tend to be stable.

To avoid the uncertainly of data distribution, we repeat this process ten times, and each time, we randomly select 20 monitoring sources and divide them equally into five groups, and then find best N in accordance with MRE. Table IV shows N of each time.

The average of best N is 39.5, as revealed in Table IV. In order to reduce the calculating time while ensuring the accuracy of prediction, in this paper, we adopt a sample size of 40 to carry out follow-up time-series data prediction.

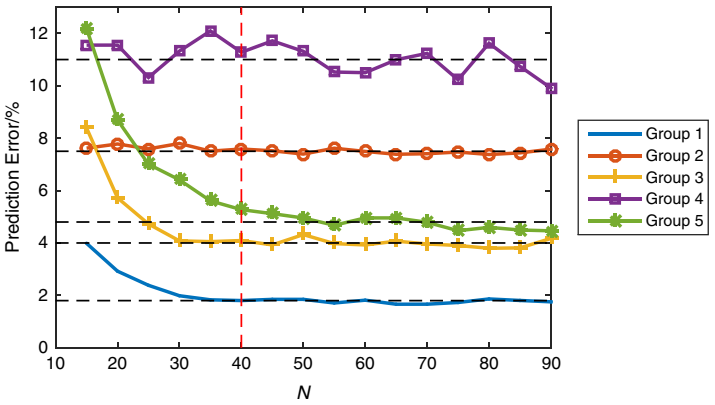


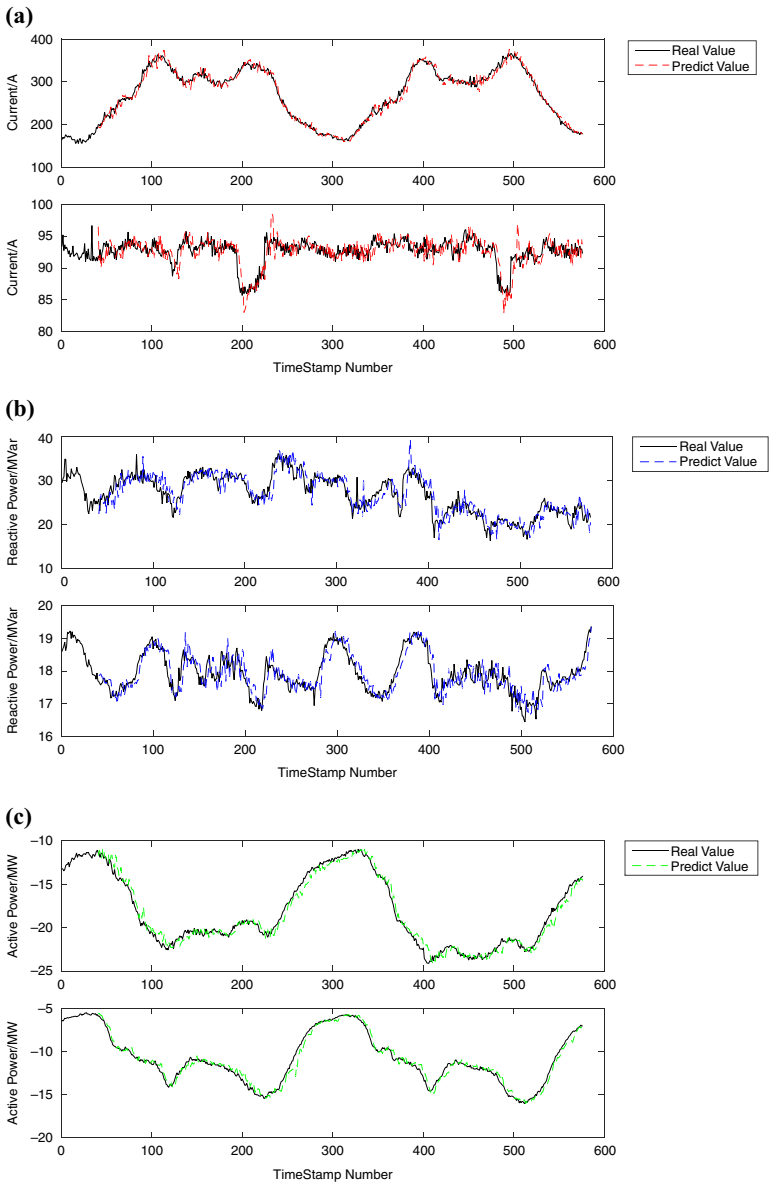
Figure 9.
Relationship between sample size N and prediction error

Table IV.
Best N of each time

No.	1	2	3	4	5	6	7	8	9	10	Avg.
Best N	40	40	35	40	45	40	35	40	35	45	39.5

5.2 Accuracy analysis

In order to demonstrate the prediction effect intuitively, we randomly select six different data sets, including the three groups: electric current, reactive power and active power, to compare disparity between prediction and actual data. The results of three groups are illustrated in Figure 10. As we can see from the figure, the predicted values are basically



Notes: (a) Electric current data; (b) reactive power data; (c) active power data

Figure 10. Three aspects prediction results of six different data sets

distributed near the actual ones; in particular, when the original data show a clear trend, e.g., active power, the predicted values are almost coincident with actual values.

In all measured sources, the numbers of active power, reactive power and electric current are 220, 180 and 100, respectively. According to the type of measured sources, prediction results are evaluated by two specific indexes, average absolute error (MAE) and average relative error (MRE), as shown in the following equations:

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}\left|\hat{y}_i - y_i\right|. \tag{5}$$

$$\mathrm{MRE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{\hat{y}_i - y_i}{y_i}\right|. \tag{6}$$

Among them, $\hat{y}_i$ means predicted value and y_i means actual value. Table V shows statistics on MAE and MRE. It can be concluded that MAE and MRE of each type are within a reasonable range.

In this experiment, the calculation of the predicted value and the comparison between forecast and actual values are performed simultaneously (shown in Figure 2). Based on previous experience or relevant statistical data, threshold for each data source is set artificially, which is displayed in Figure 11. The red line represents the threshold, the blue cross represents error between the predicted value and the actual value. The cross within the red lines indicates this is normal data. Else, if the error is larger than the threshold, it is likely to be a suspicious data, and a number of consecutive suspicious data will launch early warning. In practical applications, this design is

Table V. Prediction error statistics

Measured source type	MAE Max.	MAE Min.	MAE Avg.	MRE/% Max.	MRE/% Min.	MRE/% Avg.
Active power	3.38	0.26	1.37	7.26	1.98	3.95
Reactive power	2.32	0.11	0.98	12.51	3.54	8.51
Electric current	2.59	1.17	1.24	10.72	7.96	8.56

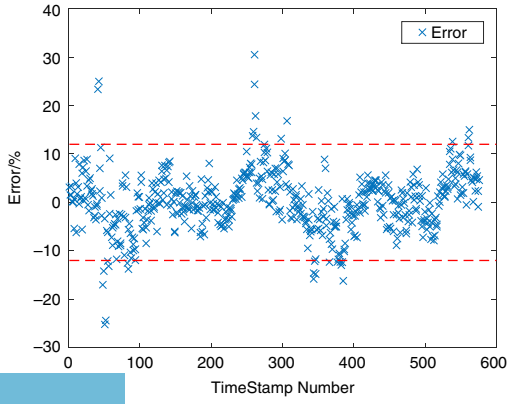


Figure 11. Real-time error analysis

helpful to reduce the workload in equipment state evaluation, especially in real-time condition-based maintenance of equipment.

A brief description should be given for the choice of the recognition method of abnormal data. High dependability on human factors makes manual selection of threshold unsustainable, and thus several related automatic methods are introduced (Marczak and Proietti, 2016). *Z*-scores, modified *z*-scores, box plots, Grubb's test, Tietjen-Moore test exponential smoothing, moving window filter algorithm, etc. are widely used methods for outliers' detection. Furthermore, the performance of the interquartile range is outstanding due to the robustness. Actually, abnormal data detection should be carried out according to the physical meaning represented by the data itself. If the outlier point represents a new data trend, it should be retained. Therefore, complementing automatic methods and manual methods with each other could be an applicable solution.

5.3 Processing efficiency of Storm

In our experiment, Spout reads the database and composes a tuple for each data source. Bolts set receives tuple and calls the corresponding ARIMA algorithm to predict results, calculate error, and store results. Times required by both Spout and Bolts sets are measured, respectively, as shown in Table VI.

As Table VI shows, Spout's running time is relatively longer than that of Bolts set, which is approximately five seconds. That is because, on one hand, Spout needs to connect the HBase and index basic information of data sources, but it may exist in different HRegion servers in HBase, and on the other hand, the performance of low-cost single-board PC (Raspberry PI) is not outstanding enough. Meanwhile, Bolts set's processing speed is very fast which is sufficient for massive data processing. Overall, compared to the time interval of measurement data acquisition (by every five minutes even longer), the processing time of Spout and Bolts sets is almost negligible, and the processing efficiency is satisfactory.

5.4 Resource consumption

During experiments, the CPU and memory usage rate of master and eight workers are monitored synchronously, as shown in Figure 12.

All eight workers' CPU usage rate ranges from 15 to 20 percent and master's usage rate is even as low as about 0.5 percent. It is because within executing process, master node is only responsible for monitoring the running status of each worker through zookeeper and assigning tasks according to workload. Concretely, master does not execute any prediction task; the specific processing work is completed by worker nodes, so the CPU usage rate of each worker is much higher than master. The situation of cluster's memory usage is quite

Group	Spout average processing time/ms	Bolts set average processing time/ms
1	4,985.1	13.0
2	4,970.4	12.2
3	4,980.4	10.9
4	5,171.7	12.3
5	5,080.3	12.6
6	5,179.9	11.8
7	5,137.8	11.8
8	5,063.9	12.8
9	5,045.0	11.7
10	5,032.7	12.7
Avg.	5,064.7	12.2

Table VI. Processing time of Spout and Bolts set

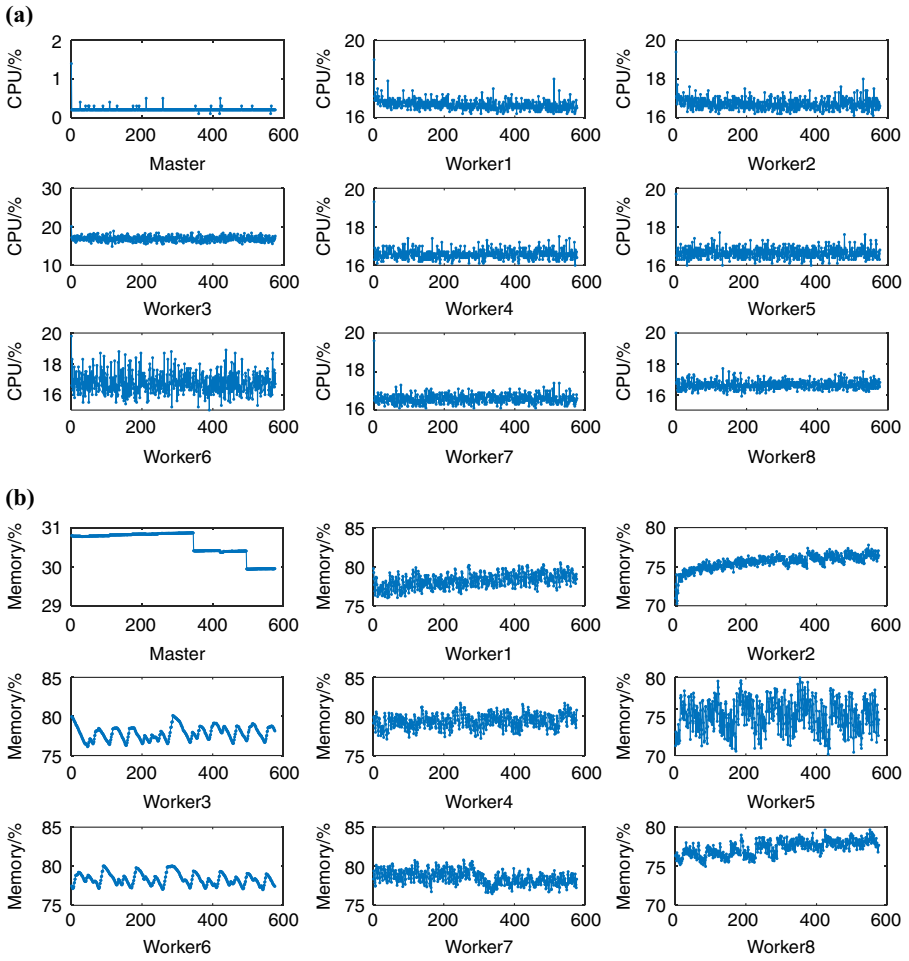


Figure 12.
Resource usage of each node

Notes: (a) CPU usage; (b) memory usage

similar with CPU usage, almost all workers' memory usage rate are more than 75 percent, but master's is even below than 31 percent. Overall, the performance of each node in the Storm cluster is rather stable, and resource consumption is relatively reasonable.

5.5 Effect of the number of nodes

We analyze the performance of this framework under different numbers of worker nodes. Since the accuracy is only related to the prediction algorithm, we focus on processing time which is spent on both Spout and Bolts sets. Speedup ratio (SR) is adopted to evaluate the parallel efficiency, as shown in the following equation:

$$SR = T_1/T_n \tag{7}$$

where T_n means the time taken by n nodes and T_1 means the time spent on a single machine.

Figure 13 shows the SR of Spout, Bolts set and ideal curve, in which we can observe that (1) SR of Spout is rather gentle, stays around 1 and almost no fluctuations occurs, and (2) SR of Bolts set is relatively optimistic, with almost linear growth at the beginning; however, with the increase of worker nodes, SR tends to be stable.

This primarily results from the properties of Spout and Bolt. As stated above, Spout is responsible for reading data, whereas Bolts set is in charge of processing data. In this experiment, the amount of data needed to read is not large, so Spout's principal working time is to establish connection with HBase and retrieval data; thus the number of nodes will not impact the running time of Spout significantly. Nevertheless, to Bolts set, the larger the number of clusters, the more nodes can be distributed to vast time-series data sources for parallel processing, and then the entire computing task will be completed faster. When the number of nodes increases further, communication and scheduling become bottlenecks that affect parallel efficiency. This is also the reason why SR tends to be smooth when seven, eight nodes occur.

5.6 Performance comparison

We implement ANN on our Storm-based platform as the prediction algorithm, and compare its performance with this proposed framework in several aspects. Specially, general regression neural network (Al-Zahrani and Abo-Monasar, 2015), a general kind of ANN, is adopted, which has an input layer, a hidden layer and an output layer. As the same with ARIMA, we use 40 sample data to predict next one; hence, the input layer holds 40 neurons. Besides, the hidden layer has typically ten neurons in this experiment.

From Sections 5.3 and 5.5, it is concluded that Spout's working time is relatively stable, no matter how many worker nodes are or which prediction algorithm is. Thus, the prediction accuracy and running time of Bolts set are chosen to compare these two solutions. In this experiment, we repeat each prediction algorithm ten times and select ten active power sources, ten reactive power sources, and ten electric current sources randomly at each time, to compare accuracy and efficiency, which are shown in Figures 14 and 15, respectively.

Two prediction algorithms, ARIMA and ANN, show no appreciable difference in accuracy, which is evaluated by MRE. The only dissenting issue is that ARIMA is a bit more stable. However, in efficiency aspect, the ARIMA algorithm shows an outstanding effect; it takes only 1/13th to 1/11th of time spent by ANN. There are two reasons which lead to this

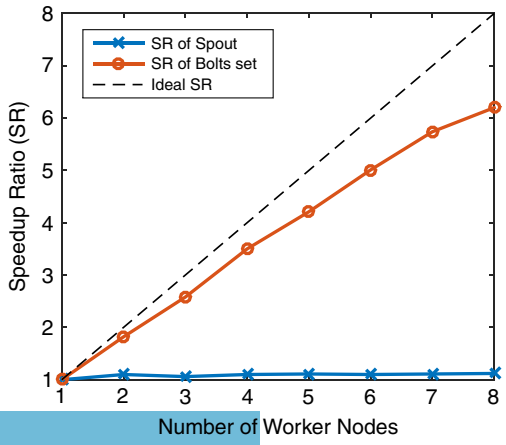


Figure 13.
SR of Spout and Bolts set

result, one is that the training process of ANN itself is complex and time consuming, and another is that the applicable model of ARIMA is prestored in the designed data structure, to a great extent, saving the training time.

6. Conclusions

In this paper, we proposed a framework based on Storm and the ARIMA algorithm to predict and analyze large-scale time-series data in real time, which includes the following characteristics:

(1) Real time and accuracy. Storm is a distributed real-time computing framework that can quickly train the ARIMA model for prediction, which ensures a fast processing speed to implement real-time processing in terms of predicting time-series data. Furthermore, the result shows that ARIMA has high accuracy in short-term prediction.

(2) High storage efficiency. A novel storage structure for time-series data is designed based on HBase, which decreases the workload and increases the use efficiency of database. Multi-version of data is able to be stored in the same cell, which facilitates the management of data quality.

(3) Parallel processing for huge amounts of data sources. It can fulfill the demand of parallel processing for vast time-series data sources with a few cluster resources by rationally allocating the number of Bolts and Spouts in Storm.

(4) Error real-time analysis and early warning. Trigger early warning once there is a persistence deviation with preset threshold, which can effectively reduce staff workload.

A proof-of-concept cluster is set to verify the performance of the proposed framework in six aspects, including the influence of sample size on the prediction result, the accuracy

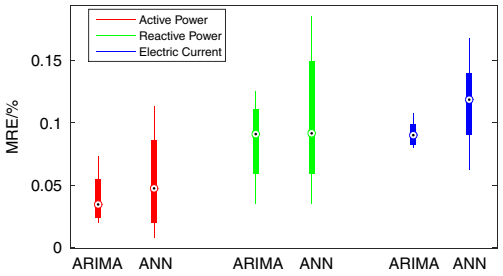


Figure 14.
Prediction accuracy comparison

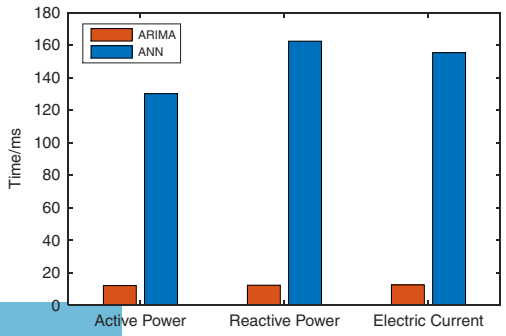


Figure 15.
Running time of Bolts set comparison

analysis, the processing efficiency of Spouts and Bolts in Storm, the resource consumption of servers, effect of the number of nodes and performance comparison between different algorithms. Experimental results demonstrate that the proposed framework of time-series data prediction based on Storm and the ARIMA algorithm achieves a rather ideal effect in terms of prediction accuracy and processing speed, which meets requirements in practical applications. Further works will focus on: optimization for Storm, including load balancing, memory optimization and cluster optimization; exploring the method to estimate better parameters of the ARIMA model; optimization for algorithm, since there is still a bottleneck on long-term forecast using ARIMA. Referring to different time series, a combination of several algorithms, such as neural network and ARIMA, is worth studying in order to improve precision of prediction.

References

Al-Zahrani, M.A. and Abo-Monasar, A. (2015), "Urban residential water demand prediction based on artificial neural networks and time series models", *Water Resources Management*, Vol. 29 No. 10, pp. 3651-3662.

ASF (2016a), "Apache Hadoop", available at: http://hadoop.apache.org/ (accessed September 13, 2016).

ASF (2016b), "Apache HBase", available at: http://hbase.apache.org/ (accessed September 13, 2016).

ASF (2016c), "Apache Spark Streaming", available at: http://spark.apache.org/streaming/ (accessed September 13, 2016).

ASF (2016d), "Apache Storm", available at: http://storm.apache.org/ (accessed September 13, 2016).

Bose, A. (2010), "Smart transmission grid applications and their supporting infrastructure", *IEEE Transactions on Smart Grid*, Vol. 1 No. 1, pp. 11-19.

Box, G.E.P., Jenkins, G.M. and Reinsel, C. (2013), *Time Series Analysis: Forecasting and Control*, 3rd ed., Wiley, Hoboken, NJ.

Chen, D., Kalra, S., Irwin, D., Shenoy, P. and Albrecht, J. (2015), "Preventing occupancy detection from smart meters", *IEEE Transactions on Smart Grid*, Vol. 6 No. 5, pp. 2426-2434.

Cheng, C., Sa-Ngasoongsong, A., Beyca, O., Le, T., Yang, H., Kong, Z. and Bukkapatnam, S.T.S. (2015), "Time series forecasting for nonlinear and non-stationary processes: a review and comparative study", *IIE Transactions (Institute of Industrial Engineers)*, Vol. 47 No. 10, pp. 1053-1071.

Cui, H., Keeton, K., Roy, I., Viswanathan, K. and Ganger, G.R. (2015), "Using data transformations for low-latency time series analysis", HP Laboratories Technical Report, No. 74, Palo Alto, CA.

Dietrich, B., Goswami, D., Chakraborty, S., Guha, A. and Gries, M. (2015), "Time series characterization of gaming workload for runtime power management", *IEEE Transactions on Computers*, Vol. 64 No. 1, pp. 260-273.

Drusinsky, D. (2016), "Run-time monitoring using bounded constraint instance discovery within big data streams", *Innovations in Systems and Software Engineering*, Vol. 12 No. 2, pp. 141-151.

EIA, US (2014), "How many smart meters are installed in the US and who has them?", Frequently asked questions, available at: www.eia.gov/tools/faqs/faq.cfm?id=108&t=3 (accessed September 13, 2016).

Fonseca, I., Farinha, T. and Barbosa, F.M. (2008), "On-condition maintenance of wind generators – from prediction algorithms to hardware for data acquisition and transmission", *WSEAS Transactions on Circuits and Systems*, Vol. 7 No. 9, pp. 909-918.

Fu, T.-C. (2011), "A review on time series data mining", *Engineering Applications of Artificial Intelligence*, Vol. 24 No. 1, pp. 164-181.

Kejariwal, A., Kulkarni, S. and Ramasamy, K. (2006), "Real time analytics: algorithms and systems", *3rd Workshop on Spatio-Temporal Database Management, Co-located with the 32nd International Conference on Very Large Data Bases, Association for Computing Machinery, Seoul, September 11*, pp. 2040-2041.

Keogh, E. and Smyth, P. (1997), "A probabilistic approach to fast pattern matching in time series databases", *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining , AAAI Press, Newport Beach, CA*, pp. 24-30.

Levenberg, A., Simpson, E., Roberts, S. and Gottlob, G. (2013), "Economic prediction using heterogeneous data stremas from the world wide web", *3rd International Workshop on Scalable Decision Making: Uncertainty, Imperfection, Deliberation, Springer International Publishing, Prague, September 23*, pp. 1-11.

Lian, X. and Chen, L. (2008), "Efficient similarity search over future stream time series", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 20 No. 1, pp. 40-54.

Liu, Q., Dong, M. and Peng, Y. (2012), "A novel method for online health prognosis of equipment based on hidden semi-Markov model using sequential Monte Carlo methods", *Mechanical Systems and Signal Processing*, Vol. 32, pp. 331-348.

Liu, Y., Choudhary, A., Zhou, J. and Khokhar, A. (2006), "A scalable distributed stream mining system for highway traffic data", *10th European Conference on Principles and Practice of Knowledge Discovery in Databases, Springer Verlag, Berlin, September 18*, pp. 309-321.

McHann, S.E. Jr (2013), "Grid analytics: how much data do you really need?", *57th IEEE Rural Electric Power Conference, Institute of Electrical and Electronics Engineers Inc., Stone Mountain, GA, April 28-May 1*, pp. 1-4.

Marczak, M. and Proietti, T. (2016), "Outlier detection in structural time series models: the indicator saturation approach", *SSRN Electronic Journal*, Vol. 32 No. 1, pp. 180-202.

Policker, S. and Geva, A.B. (2000), "A new algorithm for time series prediction by temporal fuzzy clustering", *Proceedings of 15th International Conference on Pattern Recognition, IEEE Computer Society, Los Alamitos, CA, September 3-7*, pp. 728-731.

RaspberryPIFoundation (2015), "RaspberryPI", available at: www.raspberrypi.org/ (accessed September 13, 2016).

Shibuya, T., Harada, T. and Kuniyoshi, Y. (2009), "Causality quantification and its applications: structuring and modeling of multivariate time series", *15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Association for Computing Machinery, Paris, June 28-July 1*, pp. 787-795.

Trilles, S., Schade, S., Belmonte, O. and Huerta, J. (2015), "Real-time anomaly detection from environmental data streams", *18th AGILE International Conference on Geographic Information Science, Kluwer Academic Publishers, Lisbon, June 9-June 12*, pp. 125-144.

Velasquez-Henao, J.D., Franco-Cardona, C.J. and Olaya-Morales, Y. (2012), "A review of DAN2 (dynamic architecture for artificial neural networks) model in time series forecasting", *Ingenieria y Universidad*, Vol. 16 No. 1, pp. 135-146.

Wagner, N., Michalewicz, Z., Schellenberg, S., Chiriac, C. and Mohais, A. (2011), "Intelligent techniques for forecasting multiple time series in real-world systems", *International Journal of Intelligent Computing and Cybernetics*, Vol. 4 No. 3, pp. 284-310.

Zhao, Z., Ding, W., Wang, J. and Han, Y. (2015), "A hybrid processing system for large-scale traffic sensor data", *IEEE Access*, Vol. 3, pp. 2341-2351.

Zhu, Y. and Shasha, D. (2002), "StatStream: statistical monitoring of thousands of data streams in real time", *International Conference on Very Large Data Bases*, pp. 358-369.

About the authors

Kehe Wu received the MSc Degree in Mechatronics and the PhD Degree in Computer Science from the North China Electric Power University (NCEPU), Beijing, China, in 1995 and 2005, respectively. He is currently a Professor of Information Security with NCEPU. Professor Wu is the Director of the Chinese Association for Artificial Intelligence and a Senior Member of China Electric Power Information Standardization Committee.

Yayun Zhu received the Bachelor's Degree in Computer Science and Technology from the North China Electric Power University (NCEPU), Beijing, China, in 2011. He is now pursuing the PhD Degree in Information Security at the NCEPU, Beijing, China. His research interests include Big Data in electric power and energy internet. Yayun Zhu is the corresponding author and can be contacted at: zyyaaa@126.com

Quan Li is a Graduate Student in the North China Electric Power University. His research interests include machine learning and distributed application design.

Ziwei Wu is a Graduate Student in the North China Electric Power University. Her major is computer system architecture.