Journal of Computers Vol. 29 No. 4, 2018, pp. 185-202
doi:10.3966/199115992018082904015

Improving Fault Diagnosis Performance Using Hadoop MapReduce for Efficient Classification and Analysis of Large Data Sets

Ameen Alkasem*, Hongwei Liu, Muhammad Shafiq

School of Computer Science and Technology, Harbin Institute of Technology,
92 West Dazhi Road, Nan Gang District, Harbin 150001, China
77ameen@ftcl.hit.edu.cn, {liuhw,muhammadshafiq}@hit.edu.cn

Received 7 April 2017; Revised 27 July 2017; Accepted 9 August 2017

Abstract. Underpinning a significant amount of the mass quantities of data, virtualization technology is a key element of utility cloud and an area in which monitoring is a special challenge. The monitoring of large, complex systems requires high accuracy, low latency, and near-real-time fault detection and anomaly analysis along with optimization enactment and actions for corrections. For this paper, we investigated a fine-grained fault-tolerance mechanism with newly proposed algorithms for the analysis of large datasets that are based on the Hadoop MapReduce platform, and we implement a Naïve Bayes Classifier (NBC) algorithm with Hadoop MapReduce to achieve high-performance and efficient classification for the analysis procedure that occurs in virtualization and utility cloud. Evaluation results show that the accuracy of our proposed method using Hadoop MapReduce approaches 89.80% as the size of the data sets increases. We demonstrate that our model is scalable to large data sets of virtual machine (VM) component utilization metrics with increased accuracy, low latency, and machine learning ability.

Keywords: fault diagnosis, Hadoop MapReduce, naïve bayes classifier, utility cloud, virtualization

1 Introduction

The most important feature of the utility cloud is manage of utility computing not as a product but rather as a service which that shared software, information and resources are afforded to computers, tablets, phones, etc. It is a utility delivered via a network such as the Internet. Because of the utility cloud, big data and the prevalence of computing have become increasingly important as a result data have become more valuable than ever before [1]. It is crucial to the upkeep of such massive-scale systems is monitoring for the detection of faults, errors, and anomalies for optimization enactment and so that corrective measures can be taken. The monitoring of large-scale systems is a considerable challenge, and it must include low-latency movement of huge quantities of data and real-time analysis [2]. Additionally, monitoring is an indispensable aspect of any fault diagnosis system. The data received from monitoring are very valuable, providing opportunities for the detection of faults, misconfigurations, and other noteworthy events [2-3].

However, the scale needed for the largest data set monitoring frameworks [5] (e.g., near-real-time monitoring and analysis of large data set collections) can be hundreds of nodes. Therefore, at this scale two major problems are detection and diagnosis of faults which must be rapid. Accelerated fault detection and diagnosis are even more important and are required in cloud computing services [6]: whereas a 20-second delay to recover from a fault or anomaly is a nuisance in a service provider, it can mean losing the chance to make a key decision in service recovery. More precisely, machine learning technologies such as Bayesian networks (BNs) [7] are used extensively for the classification and analysis

* Corresponding Author

of data sets because of their "learning" capabilities from the combination of data sets based on training and testing in order to make relatively highly accurate real-time decisions [8-9]. New applications and technologies have emerged in big data analysis (such as Hadoop MapReduce [10-11], Nagios [12], OpenStack [13], Aneka [14], and Weka software [15]). Weka is a machine learning workbench that supports many activities of machine learning practitioners. Recently, there has been much research in this area. Saswade et al. [16] developed an application to monitor health and performance of their critical applications deployed on Amazon Web Services (AWS). Ward and Barker [1] are proposed Varanus, a cloud-aware monitoring tool that provides robust fault-tolerant monitoring at scale. Aceto et al. [17] provided a survey of cloud monitoring; in this survey, they identified open issues, main challenges, and future directions in the field of cloud monitoring. Kumaret al. [18] developed a MapReduce framework for automatic pattern recognition based on fault diagnosis by solving a data imbalance problem in a cloud-based manufacturing (CBM) system. Smith et al. [19] proposed an availability model that combines a high-level fault tree model with a number of lower level Markov models of a blade server system (IBM BladeCenter). However, a comprehensive analysis indicates that these researchers who work in different research areas did not take into account other effects such as efficiency, accuracy, and risks in the monitoring and classification of large-scale data sets to accelerate diagnosis of faults and anomalies in cloud services.

The contribution of this paper is a proposed method based on the Hadoop MapReduce platform and the accelerated fault diagnosis innovation (AFDI) model proposed and developed in our work [20] to evaluate the scalability of large-scale data set monitoring, classification, and analysis. The method proposed uses a technique that combines two hybrid models, a Naïve Bayes Classifier (NBC) [21] and Apache Mahout [22], to accomplish a fine-grained fault-tolerance workflow based on Hadoop MapReduce and generates the highest efficacy and cost-saving fault diagnosis through three simple steps: (I) Monitoring, (II) Data pre-processing, and (III) Diagnosis and analysis, as depicted in Fig. 1. The case study used in this work is the failure of a host server to start up. Results show that running the new model across Hadoop can save a considerable amount of time compared with running the model without a Hadoop cluster without sacrificing the classification accuracy and can optimize performance analytically. The results are promising as the accuracy of the proposed method is an improvement approaching 89.80% as the size of the datasets increases.

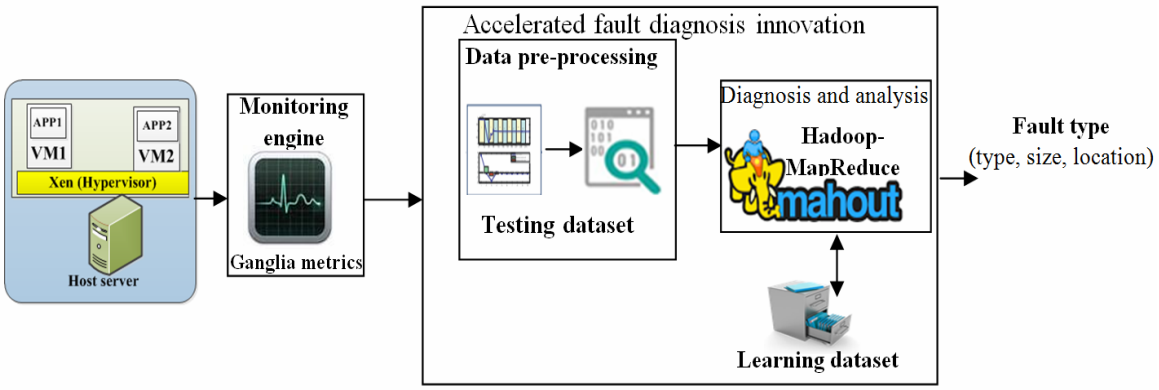


Fig. 1. Overview of a fine-grained fault-tolerance workflow based on Hadoop MapReduce

This paper is organized as follows. In Section 2, we describe the tools used in an empirical evaluation of our methodology. Section 3 focuses on materials and methods. In Section 4, we explain the experimental setup and results and provide a discussion. Finally, our conclusions and ideas for future work are given in Section 5.

2 Tools Used in Empirical Evaluation of Methodology

2.1 Hadoop MapReduce

Virtualization has emerged as a key enabling technology for cloud infrastructures. It enables multiple virtual machines (VMs) to be consolidated on a single physical machine (PM) [23]. Unfortunately, the running performance of VMs on cloud infrastructure platforms is unpredictable. This is because of complications in fault diagnosis and repair actions due to the increased complexity and scalability of monitoring data from multiple VMs consolidated on a subset of host server machines [24]. Hadoop MapReduce is a parallel programming model for large-scale data processing that can be utilized to process massive amounts of data stored in Java's Hadoop Distributed File System (HDFS) [3]. Hadoop is the open-source implementation of MapReduce. This paradigm is tightly coupled with the large data sets phenomenon. Fig. 2 illustrates the generic Hadoop MapReduce framework.

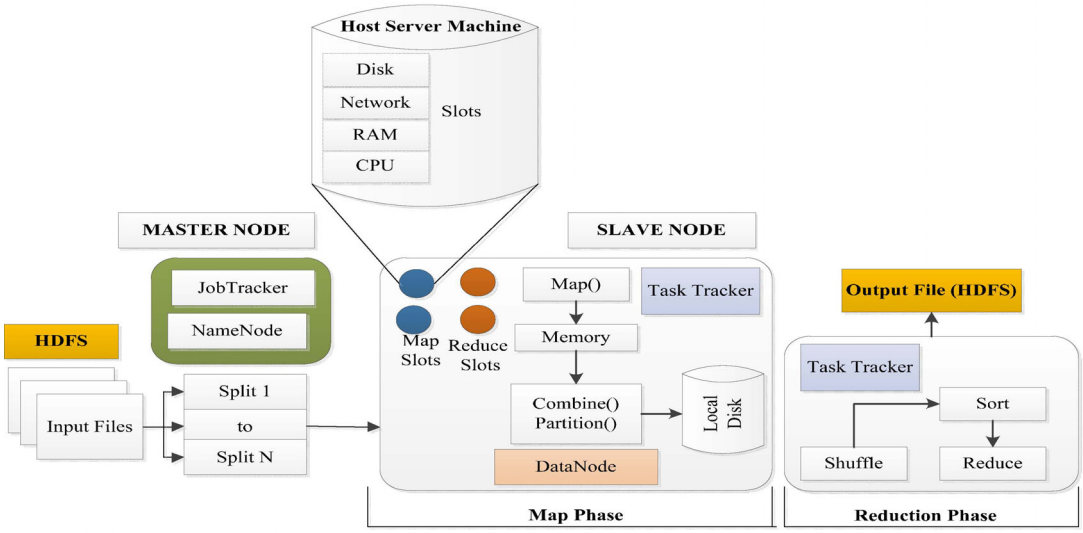


Fig. 2. Hadoop MapReduce framework

The Hadoop MapReduce framework consists of two components [25-26]. The first one is MapReduce and second one is HDFS. It adopts a master–slave architecture whereby a single master node runs the software daemons NameNode and JobTracker and multiple slave nodes run DataNode and TaskTracker. In a typical MapReduce job, the framework divides the input data into multiple splits, which are processed in parallel by map tasks. The output of each map task is stored on the corresponding TaskTracker's local disk. This is followed by a shuffle step, in which the intermediate map output is copied across the network a sort step and finally the reduce step. Different modules of the Apache Foundation also support Hadoop. These include Hive for data warehousing, ZooKeeper for high-performance coordination, and Mahout for scalable machine learning and data mining [27]. The online cluster by Google processes over 30 PB of data sets each day by running an average of over 100,000 Hadoop MapReduce jobs [28]. Therefore, we selected Hadoop MapReduce to speed up the parameter learning step in NBC when the size of the input data sets increases significantly.

2.2 Apache Mahout Machine Learning

The Apache Mahout project implements certain machine learning algorithms in the area of classification, clustering, and filtering, which implemented on top of Hadoop [22]. Mahout is written in Java and provides scalable machine learning algorithms. It is the default choice for machine learning problems in which the data sets are too large for a single machine. Not all information fusion algorithms scale well in handling large data sets. Therefore, a program might run out of memory when data are continuously loaded into the memory by an algorithm. Because of this problem, it is important to research ways in which a machine learning algorithm can convert data into the Hadoop MapReduce program and ways to

create scalability in large data sets by optimization of the algorithm. Thus, this paper proposes a model that utilizes an adapted version of Hadoop MapReduce whereby the generation of a new algorithm using machine learning methods is enabled for resolving the problem of large data sets [29-30].

2.3 NBC Availability Model

The Naïve Bayes Classifier (NBC) is a simple and effective supervised machine learning algorithm that can be applied in the Hadoop MapReduce paradigm [31].The NBC model is a specific Bayesian network that is widely used for classification. Classification is one of the most important problems in machine learning and big data. Suppose there are n possible classes (predictions) $X = \{x_1, x_2, \ldots, x_n\}$ for a domain of components $A = \{a_1, a_2, \ldots, a_n\}$.Let $Y = \{y_1, y_2, \ldots, y_n\}$ be the set of unique characteristics in which each must appear a minimum of one time in one of the components in A. This is illustrated in Fig. 3. The likelihood of a component's being in a certain class is capable of being computed by the Bayes theorem [32-33], as defined in equation (1).

$$P(X_i \mid A) = \frac{P(A \mid X_i)P(X_i)}{\sum_{i=1}^{n} P(A \mid X_i)P(X_i)}, i = 1, 2, \ldots, n \quad \textbf{(1)}$$

where

$P(X_i \mid A)$ is the posterior probability of the class (target) given the predictor (attribute),

$P(X_i)$ is the prior probability of the class,

$P(A \mid X_i)$ is the likelihood probability, and

$\sum_{i=1}^{n} P(A \mid X_i)P(X_i)$ is the prior probability of the predictor (evidence).

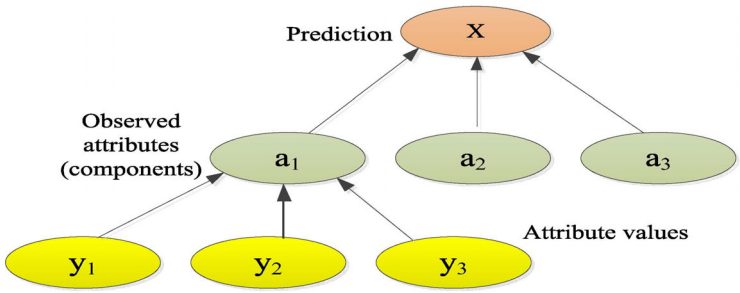


Fig. 3. Naïve Bayes model

For classification, an NBC is commonly employed because of its capability of learning from training datasets to making decisions with a new testing dataset so that it can predict salient features and deliver relatively highly accurate real-time analysis [34]. We have proposed an algorithm, shown in Fig. 4, for training and classification with the NBC model.

Input: Attribute values $y=(y_1,\ldots,y_i)$,Classes $x=(x_1,\ldots,x_i)$, $n=i$

Output: Class with highest value of estimated probability $P(X)$

1. **For** all y_i and for all classes x_j **do**
2. Calculate the conditional probability $P(y_j \mid x_i)$ as the relative frequency of y_i among those training datasets that belong to x_j
3. **End for**
4. **For** all classes x_j **do**
5. Estimate $P(x_j)$ as the relative frequency of this

---

class in the training datasets,

6. Calculate the conditional probability $P(y|x_i)$ using the naïve assumption of mutually independent attributes:

$$P(y|x_i) = \prod_{i=1}^{n} P(y_i|x_j) \quad \textbf{(2)}$$

7. **End for**

8. Choose the class with the highest value of estimate:

$$P(x_j) \times \prod_{i=1}^{n} P(y_i|x_j) \quad \textbf{(3)}$$

9. **End.**

Fig. 4. Algorithm for training and classification with NBC model

The posterior probability can be calculated by first constructing a frequency table for each attribute against the target, then transforming the frequency tables to likelihood tables, and finally using the Naïve Bayesian model to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of the prediction [35] an example is given in Table 1. This frequency will be the evaluation calculated by the proposed new model based on Hadoop MapReduce.

Table 1. Example of a frequency and likelihood table

Frequency Table		Fault State Yes	Fault State No
CPU	Normal	400	100
	Minor	500	12
	Serious	0	0

Likelihood Table		Fault State Yes	Fault State No	P(x) = P(prediction)
CPU	Normal	400/1200	100/800	500/2000
	Minor	500/1200	12/800	512/2000
	Serious	0/1200	0/800	0/2000
P(c) = P(Yes/No)		900/1200	112/800	

2.4 AFDI Model Description

The AFDI model is a new hybrid model using the advantages of the Multi-Decision Diagram (MDD) and NBC models and a virtual sensor cloud, which provides a new approach and framework for fault management in cloud services. Addressing the full life cycle of problem determination based on the severity levels and anomalies for VM/physical server metrics. The AFDI model is aimed at reducing the time and the cost of a fault diagnosis through accelerated fault diagnosis [20]. AFDI monitors a wide range of metrics across the VM and the physical server as shown in Fig. 5. In most prior work, the metrics were monitored based on the severity levels of consequences according to the symptoms [36].

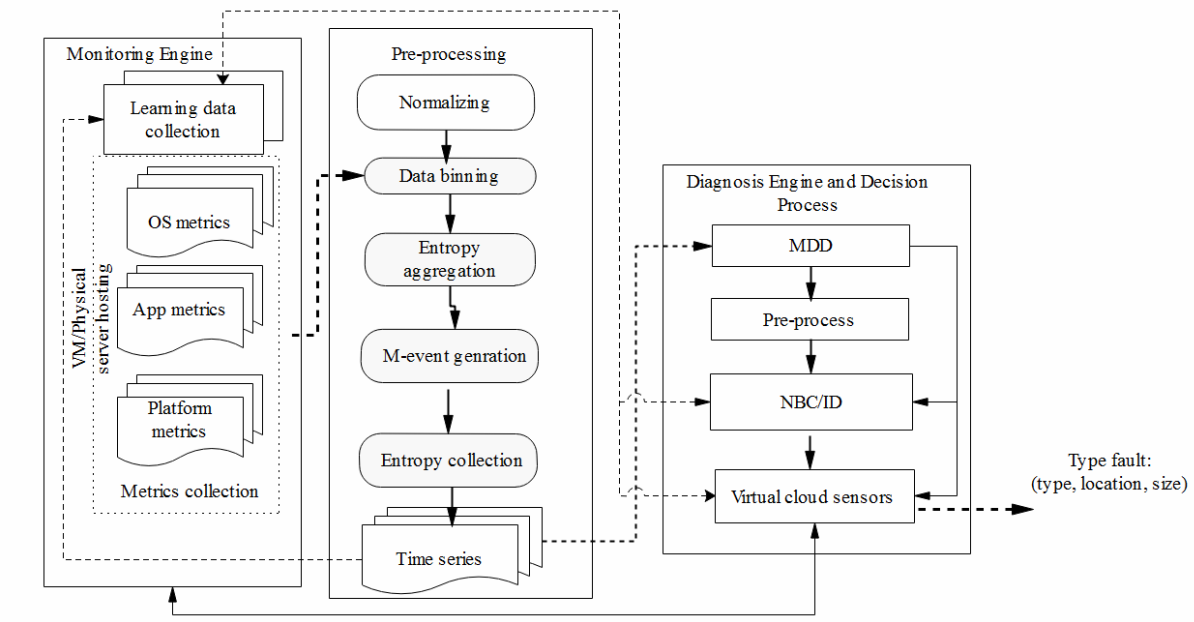


Fig. 5. AFDI monitors a wide range of metrics across the VM and the Physical server

3 Materials and Methods

3.1 Algorithms for Improving Performance

We are proposing a new model based on AFDI to evaluate whether NBC and Hadoop MapReduce can be scaled up to classify large data sets while achieving high performance for classification and analysis of metrics of large datasets. This section explains the new algorithms and the workflow of our proposed method. Illustrated in Fig. 6, the procedure includes four new modules to Hadoop along with the techniques for method evaluation (Apache Mahout):

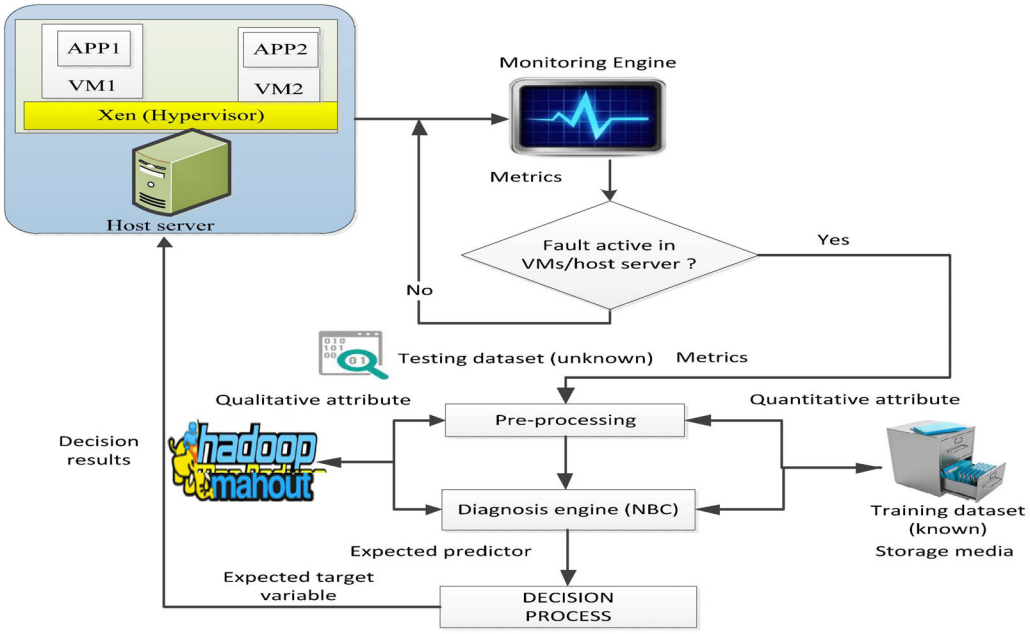


Fig. 6. Proposed framework for a fine-grained approach with Hadoop MapReduce

Monitoring engine. This module monitors each VM and host server to collect the metrics of interest (numerical predictors or attributes).
Pre-processing. All new testing data from the metrics collection monitor are normalized and binned into

intervals, and the raw time-series data are used to generate monitoring events (m-events); more detail is given in Section 3.2.

Diagnosis engine. The model's proposed methodology analyzes the time series entropy to locate patterns that signify faults or anomalies in the system being monitored.

Decision process model. A look-back window of size 3 creates a dataset table according to the following decision values:

- If (0 ≤ decision value < 2), then fault category is normal, and node fault state is normal;
- If (2 ≤ decision value ≤ 3), then fault category is minor, and node fault state is normal;
- If (decision value > 3), then fault category is serious, and node fault state is abnormal.

More details are given in Section 3.2.

In the work presented, the focus has been to enhance the AFDI model for differing sizes of data sets and to assess the Hadoop MapReduce program for its capability of learning from past data with the purpose of achieving high-performance fault diagnosis for the newly proposed model. As shown in Fig. 6, raw data are taken from the testing metric data collected by large monitoring engines (Xen-Hypervisor [37-38]) as shown in Table 2 and from a historical training dataset as shown in Table 3.

Table 2. Sample dataset of testing metrics

Time- monitoring	CPU_ Utilization	Network- overhead	Memory- usage	VM/Host state	Fault state
12:02:00 AM	30.00	30.00	30.93	?	?
12:07:00 AM	30.00	35.00	78.12	?	?
12:12:00 AM	90.00	70.00	92.43	?	?
12:17:00 AM	31.84	38.21	42.19	?	?
12:22:00 AM	27.28	32.74	36.15	?	?
⋮	⋮	⋮	⋮	⋮	⋮

Table 3. Sample historical training dataset

Time_monitoring	CPU_ Utilization	Network_overhead	Memory_usage	VM/Host_state	Fault state
12:57:00 AM	57.22	68.66	99.05	Serious	Yes
1:02:00 AM	23.34	28.01	35.02	Normal	No
1:07:00 AM	58.96	70.75	20.12	Minor	No
1:12:00 AM	69.76	83.71	10.04	Serious	Yes
1:17:00 AM	31.51	37.84	41.74	Normal	No
⋮	⋮	⋮	⋮	⋮	⋮

3.2 Pre-processing of Raw Metrics

The metrics within each look-back window are first pre-processed and transformed into a form that can be readily used by the proposed approach. With each monitoring engine sample, multiple types of metric can be collected simultaneously; e.g., in our experiment, as shown in Table 2, we collected CPU utilization, memory usage, and network overhead in each monitoring sample. Once the collection of sample data is complete, the data are preprocessed and transformed into a series of bin numbers for every metric type, using equations (4), (5), (6) to perform the data binning, with a time instance serve ($t_1, t_2, \ldots, t_i$) and m-events ($M_1, M_2, \ldots, M_i$) where i is number of instances the results as shown in Table 4 and Table 5.

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{4}$$

$$\sigma = \left[\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2\right]^{0.5} \tag{5}$$

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-1}{2}(\frac{x-\mu}{2\sigma^2})^2} \tag{6}$$

In the above equations, μ is the population mean, σ is the population standard deviation, x is from the domain of measurement values ($-\infty < x < \infty$), and n is the number of components or services. A normal probability distribution is defined by the two parameters μ and σ. Besides μ and σ, the normal probability density function (PDF) $f(x)$ depends on the constants e (approximately 2.718) and π (approximately 3.142) [39]; because these attributes are numerical data, the numerical variables in Table 3 need to be transformed into their categorical counterparts (binning) before their frequency tables are constructed by equations (7) and (8) (e.g., look-back window size = 3, range = [0, 2], and number of bins = 6), as shown in Table 4 and Table 5. The values for *binning-value* and *decision-value* are determined by the following formulas:

$$\text{IF } (x > 2) \text{ then } binning - value = 5, \text{ else } binning - value = \text{TRUNC}(x / 0.4) \quad \textbf{(7)}$$

$$decision - value = \text{MAX}(binning - value) \quad \textbf{(8)}$$

where x is the normalization value for the attribute and 0.4 is a statistic suggested by the probability values. Then can begin the classification and analysis into a dataset probability (predictor) by use of the cumulative distribution function (CDF) [39] as shown in Table 6. For a continuous random variable, the CDF equation is

$$P(X \le x) = \frac{x - a}{b - a} \quad \textbf{(9)}$$

where a is the lower limit and b is the upper limit 5, $a \le x \le b$.

Table 4. Sample of data normalization used for binning

	Time-instance	CPU	Network	Memory
Window size (3)	t_1	0.60	0.67	0.46
	t_2	0.60	0.78	1.16
	t_3	1.80	1.56	1.38
	t_4	1.05	1.15	0.97
	t_5	0.90	0.99	0.83
	t_6	1.06	0.86	1.20
	⋮	⋮	⋮	⋮

Table 5. Sample of dataset binning with decision values

	M-events	CPU	Network	Memory	Decision value
Window size (3)	M_1	1	1	1	1
	M_2	1	1	2	2
	M_3	4	3	3	4
	M_4	2	2	2	2
	M_5	2	2	2	2
	M_6	2	2	3	3
	⋮	⋮	⋮	⋮	⋮

Table 6. Sample of the classifier probability (predictor) dataset

Process Instances	CPU	Network	Memory	VM/Host state	Fault state
Process.inst1	0.20	0.20	0.20	Normal	?
Process.inst2	0.20	0.20	0.40	Minor	?
Process.inst3	0.80	0.60	0.60	Serious	?
Process.inst4	0.40	0.40	0.40	Minor	?
Process.inst5	0.40	0.40	0.40	Minor	?
Process.inst6	0.40	0.40	0.60	Minor	?
⋮	⋮	⋮	⋮	⋮	⋮

3.3 Diagnosis and Classifier Engine

3.3.1 Diagnosis Approach

The proposed diagnosis approach is a general concept that represents a combination of analytical tools cooperating to form the analysis and diagnosis model shown in Fig. 4. Our raw data were taken from large engines monitoring a host server and collected by research communities. A monitoring engine collected and processed measurement data such as basic resource metrics for each VM and host server, as listed in Table 7.

Table 7. System and application metrics monitored in new model

System metric	Description	Measurement level
CPU utilization	% CPU time in user-space/kernel-space	VM, Host
Memory utilization	% of memory used	VM, Host
Network bandwidth	% of network bandwidth	VM, Host
I/O storage	% of disk throughput	Host

The diagnosis engine contains the hybrid intelligent models MDD and NBC. The MDD is used to quickly obtain the overall fault severity level and if necessary generate an alarm in a virtual sensor cloud. The MDD processes and analyzes m-events and executes its diagnosis by pre-processing and the use of I/O pairs to calculate the parameters estimated by NBC in Hadoop MapReduce for large data sets. The NBC measures are given as a 2 by2 contingency table [40-41], shown in Table 8.

Table 8. Contingency table of measures for analysis

Prediction	Class	Observation	
		Yes	No
	Fault	True positive (TP)	False positive (FP)
	Not fault	False negative (FN)	True negative (TN)

We utilized three statistical measures, recall, precision, and accuracy, to evaluate the effectiveness of fault diagnosis using NBC in Hadoop MapReduce for large-data-set problems. More details and results are given in Section 4. The data parser first pre-processes the metrics for all components recorded in a common XML file [42]. After pre-processing, each file contains two parts: the attributes of the components and the data, as shown in Fig. 7 (a) to Fig. 7(c). Each observation of metrics for components is recorded as a single line of data, where the first three columns represent the metrics of attribute components and the last two columns represent the component state class(normal, minor and serious) and the system fault state class(yes and no) as labeled. Each column is separated by a comma. This is useful because by default Hadoop MapReduce splits the input files byline and passes each line to a mapper function to generate informational data used for the frequency table, as shown in Fig. 7(d). All pre-processed component metrics are stored in the master node as a repository while waiting for further sampling. Therefore, the proposed approach maintains a buffer size of 3 (a look-back window) of the last *n* samples' observed metrics. The metrics observed in the look-back window each time instance serve as inputs to be pre-processed. There are multiple reasons that use of a look-back window can further increase detection performance:

- At exascale, it is impractical to maintain all historical data.
- Shifts in work patterns may render old history data useless or even misleading.
- The look-back window can be implemented in high-speed RAM.

```
@relation Testing_Dataset

@attribute CPU_Utilization real
@attribute Memory_Usage
@attribute Network_Overhead real
@attribute VMs-Host_State {Normal,Minor,Serious}
@attribute Fault_State{Yes,No}

@data
1 30.00,30.30,93.00,?,?
2 30.00,35.00,78.12,?,?
3 90.00,38.21,42.19,?,?
4 27.28,32.74,36.15,?,?
5 .       .
6 .       .
7 .       .
```

(a) Testing data set (unknown)

```
@relation Learning-Dataset

@attribute CPU_Utilization real
@attribute Memory_Usage
@attribute Network_Overhead real
@attribute VMs-Host_State {Normal,Minor,Serious}
@attribute Fault_State{Yes,No}

@data
999990 47,32,13,Minor,no
999991 16,75,15,Serious ,yes
999992 50,36,14,Minor,no
999993 12,15,14,Normal,no
999994 19,20,14,Normal,no
999995 30,40,10,Minor,no
999996 26,75,22,Serious ,yes
999997 67,26,12,Minor,no
999998 19,23,19,Normal,no
999999 23,22,13,Normal,no
1000000 22,11,24,Normal,no
```

(b) Training data set (known)

```
@relation Testing_Dataset

@attribute CPU_Utilization real
@attribute Memory_Usage
@attribute Network_Overhead real
@attribute VMs-Host_State {Normal,Minor,Serious}
@attribute Fault_State{Yes,No}

@data
1 30.00,30.30,93.00,Normal,No
2 30.00,35.00,78.12,Minor,No
3 90.00,38.21,42.19,Serious,Yes
4 27.28,32.74,36.15,Minor,No
5 .       .
6 .       .
7 .       .
```

(c) Testing data set after classification and training (known)

```
1  Component_state  Frequency
2  CPU-Minor-no------396825
3  CPU-Normal-no-----402117
4  CPU-Serioues-yes--201058
5  Memory-Minor-no--396825
6  Memory-Normal-no--402117
7  Memory-Serious-yes---201058
8  network-Minor-no-------396825
9  network-Normal-no------402117
10 network-Serious-yes---201058
11 State_component-Minor-no--396825
12 State_component-Normal-no--402117
13 State_component-Serious-yes--201058
```

(d) Informational data used for the frequency table

Fig. 7. Dataset modeling (XML format)

3.3.2 Classifier Approach

The classification of component states and system states is the key step in the workflow. Fig. 6 shows the job sequence of this step. When the training data and testing data are ready in HDFS, AFDI starts the training job to build a model. The algorithm combines the test data with the model and generates an intermediate table. Finally, the algorithm classifies the job and simultaneously computes the probability of each component's being in each of the three classes and then makes a decision about the final system state. Statistics regarding the contingency table and intermediate values are recorded. In our experiments, we used two data sets: (1) a new testing dataset (unknown) and (2) a training dataset (known) as shown in Table 2 and Table 3, which convert these data to XML format to input to the algorithms of the proposed method. The formation of XML files, shown in Fig. 7(a) to Fig. 7(b) and Fig. 7(d), is our simple dataset framework for a state-based probability model that predicts component utilization. In Fig. 7, the measurement level varies from 0 to 100%. The component utilization uses CPU $\in$ {0-25%, 26%-75%, 76%-100%} as thresholds. We observed the percentage component utilization at discrete times $t_1, \ldots, t_n$.

The new dataset holds the model classifier results given by combining the testing and training datasets, as shown in Fig. 7(c). We first define our task to classify the three state components using an NBC model. The system cannot work if one component is faulty. We use 0, 1 and 2 to represent the system and component states, where 0 denotes good status (normal working conditions), 1 denotes a minor fault and 2 denotes a serious fault. For example, CPU, memory and network represent the three basic components and host-server-state represents the system state as shown in Fig. 8. Hence, we have only three classes (normal, minor and serious) for the component state measurements and only two classes (yes and no) for

the system fault state. To simplify the problem, we choose the same number of normal, minor, and serious measures for the components. The classification problem is then converted to a counting problem on the training and testing datasets.

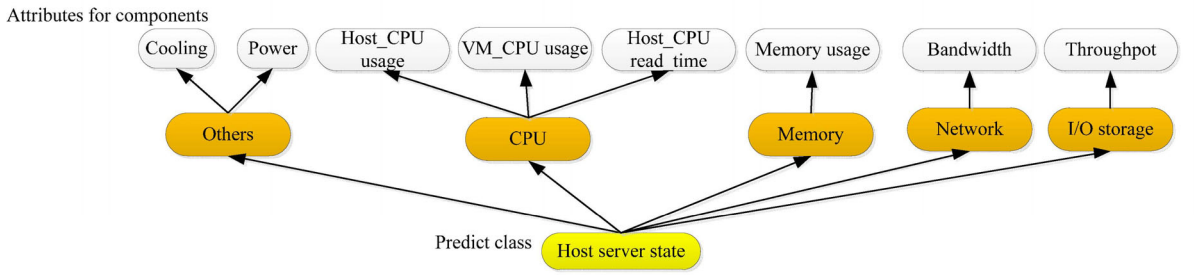


Fig. 8. Model predicting component utilization for host server

We divided the problem solving into two algorithms:

(1) An algorithm for the training and filtering, shown in Fig. 9. In this algorithm, all training instances are fed into the job to produce a model for all instant attributes, and the data set is filtered with their frequencies of normal, minor, and serious component states.

```
Input: value←Training datasets. //XML format
Output: Result. // reduce result save in HDFS
1- Begin
2- Map(Object key, Text value, Context context)
3-    component_name ←"";
4-      Row[]← value.toString().split(",");
5-        if(row.length>= 5)
6-         state_component←row[3];
7-          for col=0 to col<=3 do  // filtering datasets
8-            component_name ← "State_component"
9-             elseif (row[col]>=0&&row[col] <=25) state_component="Normal";
10-              elseif (row[col]>=26&&row[col] <=74) state_component="Minor";
11-              elseif (row[col]>=75&&row[col] <=100) state_component="Serious";
12-               if (col==0) component_name="CPU";
13-               elseif (col==1) component_name="Memory";
14-               elseif (col==2) component_name=" Network"
15-                word.set(component_name+"-"+state_component+"-"+row[4]);
16-                context.write(word, one);
17-           End for
18-      End if
19-   End Map
20- Reduce(Text key, Iterable<IntWritable>values, Context context)
21-       Sum←0;
22-            for (Int Writableval:values) do
23-               sum += val.get();
24-            End for
25-            result.set(sum);
26-              context.write(key, result);
27- End reduce
28- Result←result;
29- End
```

Fig. 9. Algorithm for training and filtering data sets

(2) Combining the test and training data set algorithm, as shown in Fig. 10. In this algorithm, the training and the test data set measures are combined to give an intermediate table with all information necessary for the final classification. This algorithm classifies all instances to only two final fault state classes (yes, no) and writes the final resulting classification to Hadoop (HDFS).

```
Input:  Training and testing datasets //XML format
Output: A new-dataset-result.XML, F.Measure, Precision and Recall.
1- Begin
2-    Classify(Dataset-train, Dataset-test) //Datasets with XML format
3-       BufferedReader breader←null;
4-         Breader← new BufferedReader(new FileReader(Dataset-train));
5-           Instances train←new Instances(breader);
6-              train.setClassIndex(train.numAttributes()-1);
7-                 Breader←new BufferedReader(newFileReader(Dataset-test));
8-                Instances test←new Instances (breader);
9-                      test.setClassIndex(train.numAttributes()-1);
10-                    breader.close();
11-                     J48 tree←new J48();
12-                           tree.buildClassifier(train); //build classifier
13-                   Instances labeled← new Instances(test);//label instances
14-                   for i=0 to i<test.numInstances()do
15-                      clsLabel←tree.classifyInstance(test.instance(i));
16-                      labeled.instance(i).setClassValue(clsLabel);
17-                   End for
18-                        BufferedWriter writer←new BufferedWriter(new
                       FileWri ter("Anew-dataset-result.XML"));
19-                      writer.write(labeled.toString());
20-                    writer.close();
21-                      Call-Produce StartNBC("A new-dataset-result.XML");
22-    End //classify
23-      StartNBC (Dataset)
24-       BufferedReader breader ←null;
25-    breader ←new BufferedReader(new FileReader(Dataset));
26-       Instances train←new Instances (breader);
27-             train.setClassIndex(train.numAttributes()-1);
28-            breader.close();
29-             NaiveBayesnB←new NaiveBayes();
30-                  nB.buildClassifier(train);
31-                 Evaluation eval←new Evaluation(train);
32-                      eval.crossValidateModel(nB, train, 20, new
   Random(1));
33-                         The F.Measure← eval.fMeasure(1);
34-                           The Precision← eval.precision(1);
35-                              The Recall← eval.recall(1)
36- End
```

Fig. 10. Algorithm for combining the test and training data sets

After these two algorithms are finished, the results collector retrieves the model classification results intermediate values table and test data statistics from HDFS's Hadoop storage [43]. By the end of the classification analysis algorithms. All instances have been classified into a yes/no final fault state with normal, minor, and serious state classes as shown in Table 9 and Fig. 11.

Table 9. Sample of the dataset classifier probability results

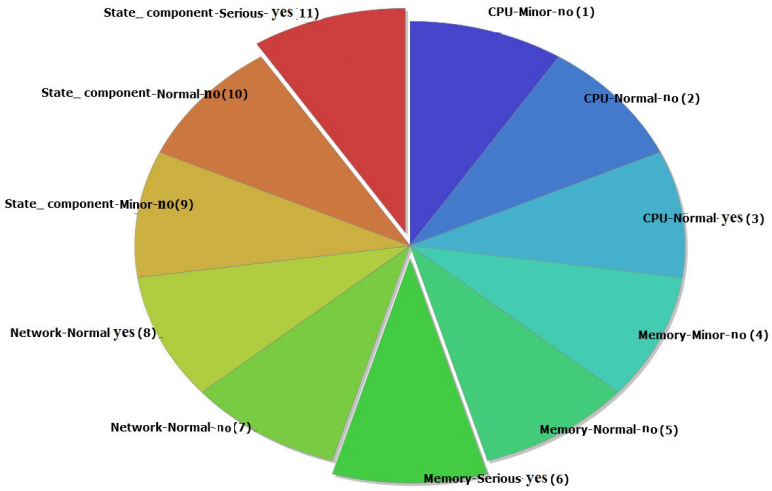
Process. Instance	CPU	Network	Memory	VMs/Host state	Fault state
Process.inst1	0.25	0.25	0.25	Normal	No
Process.inst2	0.25	0.25	0.50	Normal	No
Process.inst3	1.00	0.75	0.75	Serious	Yes
Process.inst4	0.50	0.50	0.50	Minor	No
Process.inst5	0.50	0.50	0.50	Minor	No
Process.inst6	0.50	0.50	0.50	Minor	No
⋮	⋮	⋮	⋮	⋮	⋮

```
7999992 50,36,14,Minor,no
7999993 12,15,14,Normal,no
7999994 19,20,14,Normal,no
7999995 47,32,13,Minor,no
7999996 16,75,15,Serious,yes
7999997 50,36,14,Minor,no
7999998 12,15,14,Normal,no
7999999 19,20,14,Normal,no
8000000 12,15,14,Normal,no
```

(a) Testing data set (known) after diagnosis and analysis

```
1  CPU-Minor-no    3174600
2  CPU-Normal-no   3216936
3  CPU-Normal-yes  1608464
4  Memory-Minor-no 3174600
5  Memory-Normal-no    3216936
6  Memory-Serious-yes  1608464
7  Network-Normal-no   6391536
8  Network-Normal-yes  1608464
9  State_component-Minor-no    3174600
10 State_component-Normal-no   3216936
11 State_component-Serious-yes 1608464
```

(b) Hadoop's HDFS result



(**c**) Pie chart showing the relative frequencies of the HDFS final dataset results

Fig. 11. Implementation of approach with new algorithms

4 Experiment Setup

The experiment setup used two VMs (VM1 and VM2) on a Xen-Hypervisor platform hosted on one Dell blade server with dual-core 3.9 GHz CPUs and 4 GB RAM, as shown in Fig. 12.

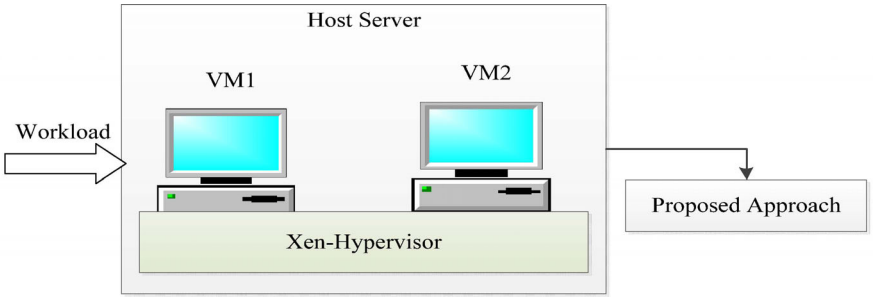


Fig. 12. Architecture of the experiment system setup

We implemented our approach at both the VM level and the host server level. The host server is considered the direct parent of the VMs. Thus, the local time series is calculated first for the VMs followed by their aggregation for the host server's global time series. The first implementation uses Xen-Hypervisor and the Ganglia metrics method to record and identify global resource provisioning anomalies. Forty anomaly samples were injected into the testbed leading to global resource consumption by the anomalies/faults, which do not exclude the CPU utilization of the running host server. The VM metrics and the host metrics were collected using the Xen-Hypervisor and analyzed in a fault detector and classifier. To test the scalability and performance of the new model, the dataset size was varied from 1,000,000 to more than 32,000,000 instances of recorded metrics in each class.

4.1 Results

The resulting statistics include the classification accuracy by algorithm as shown in Fig. 13 and Fig. 14 demonstrating high accuracy for the classification of component states (normal, minor, and serious). The accuracy and throughput of the system without evaluation by Hadoop MapReduce code are shown in Fig. 15.

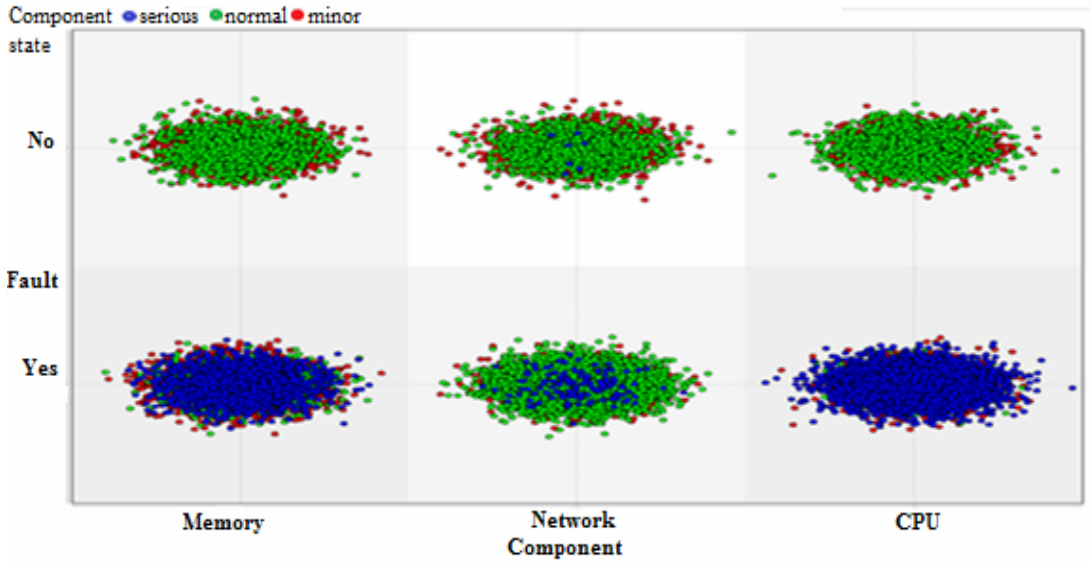


Fig. 13. Statistical analysis of the component state classification

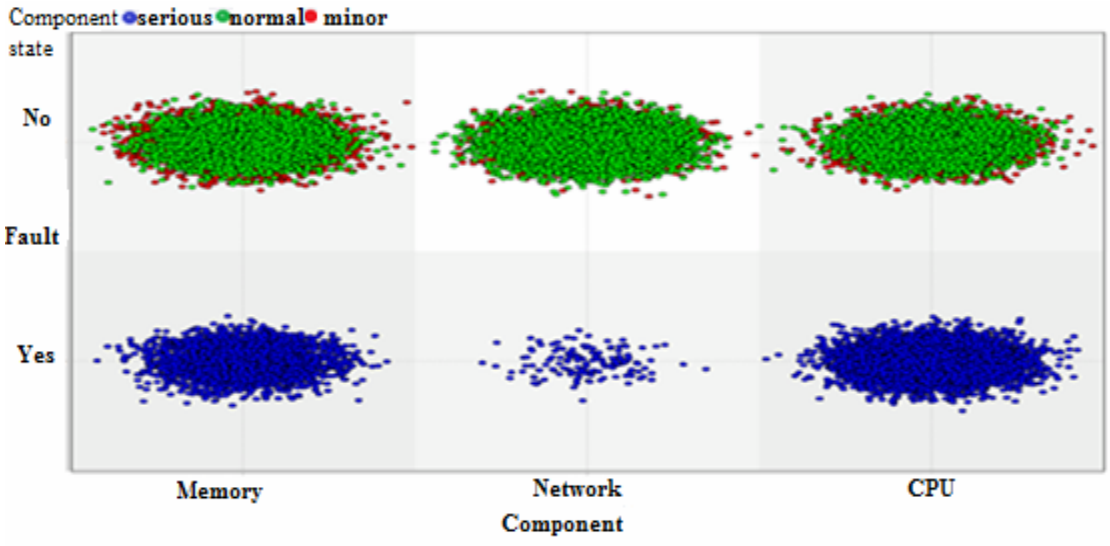


Fig. 14. Result statistics include the classification accuracy by algorithms

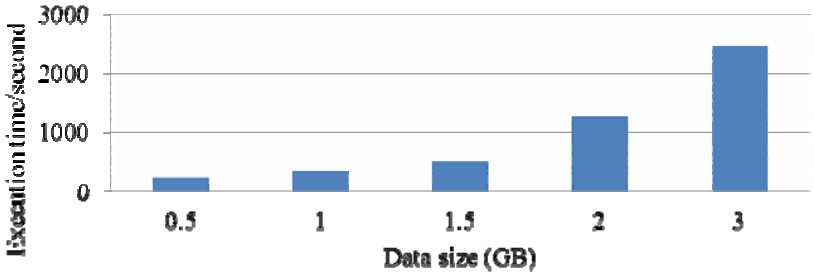


Fig. 15. Throughput of system with respect to dataset size without Hadoop MapReduce

4.2 Discussion

In a previous study, we executed our AFDI model and achieved an average accuracy of 92.1% with small data sets. That model was able to classify different subsets of the host server monitoring dataset with comparable accuracy. We used the four statistical measures given by equations (10), (11), (12), and (13) to evaluate the effectiveness of anomaly/fault detection [44].

$$Recall = \frac{TP}{TP + FN} \tag{10}$$

$$Precision = \frac{TP}{TP + FP} \tag{11}$$

$$Accuracy = \frac{TP + TN}{Precision + Recall} \tag{12}$$

$$False\ alarm\ rate\ (FAR) = \frac{(Number\ of\ \ false\ alarms)}{(Total\ number\ of\ \ alarms)} = 1 - Precision \tag{13}$$

where

TP (true positives) is the number of positive instances correctly predicted,

FN (false negatives) is the number of positive instances wrongly predicted as negative,

FP (false positives) is the number of negative instances wrongly predicted as positive, and

TN (true negatives) is the number of negative instances correctly predicted.

Our experimental results reveal several interesting findings for the evaluation of NBC in the new model with machine learning (Apache Mahout), a performance summary and results of different implementations as shown in Fig. 16. We achieved a high accuracy of up to 89.80% and 5-20% false alarm rate. The use of Hadoop MapReduce can be optimized to speed up the parameter learning stage in NBC for a wide range of input dataset sizes.

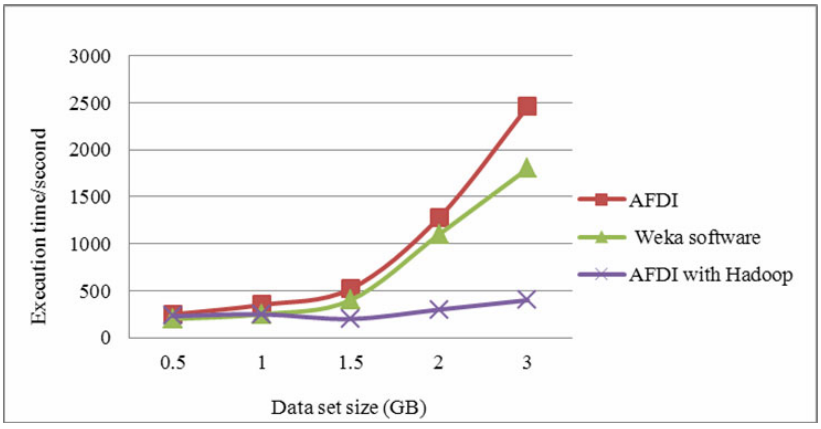


Fig. 16. Throughput of system with performance summary and results of different implementations

The number of instances the system can process and analyze in one second increased from 0.54 GB to 3 GB. A performance summary of various results is presented in Table 10.

Table 10. Summary comparison of performance results

Model	Accuracy (%)	Time(seconds) $\times 10^3$	Dataset size(GB)
AFDI	84.86	2.46	3
Weka software	85.88	1.80	3
Proposed method	**89.80**	**0.47**	3

Finally, our new approach methodology can scale up to more than 32 million instances of recorded metrics in each class from a monitoring engine with high accuracy and time cost savings compared with other models proposed.

5 Conclusion

In this paper, a new diagnosis approach that represents a combination of analytical tools to realize system analysis and diagnosis has been proposed. We presented a fine-grained fault-tolerance mechanism for algorithms for the analysis of large datasets based on the Hadoop MapReduce platform, and generates the highest efficacy and cost-saving fault diagnosis through three simple steps: (I) monitoring, (II) data pre-processing, and (III) diagnosis and analysis. The proposed model demonstrates that NBC is able to scale up to the classification of metrics for large data sets collected from monitoring VM/host servers and analyzed in near-real time with increased accuracy, low latency, and machine learning ability. The additional modules inserted to evaluate the newly proposed algorithms showed good classification and an accuracy of up to 89.80% with 5%-20% false alarm rate. We believe that our work is just the beginning for the use of machine learning technologies in the classification and analysis of large data sets. Future work will examine other intelligent algorithms with Apache Spark for a fast general engine for the processing of large data sets and the automation of fault repair and recovery actions.

Acknowledgments

This work was supported by the Chinese High Tech R&D (863) Program Project “Cloud Computer Test and Evaluation System Development (2013AA01A215).”

References

[1] J.S. Ward, A. Barker. Cloud cover: monitoring large-scale clouds with varanus, Journal of Cloud Computing 4(1)(2015) 1-28.

[2] H. Chen, R.H. Chiang, V.C. Storey, Business intelligence and analytics: from big data to big impact, MIS Quarterly 36(4)(2012) 1165-88.

[3] J. Demšar, B. Zupan, G. Leban, T. Curk, Orange: from experimental machine learning to interactive data mining, in: Proc. European Conference on Principles of Data Mining and Knowledge Discovery, 2004.

[4] R.J. Patton, P.M. Frank, R.N. Clark (Eds.), Issues of Fault Diagnosis for Dynamic Systems, Springer Science & Business Media, New York, 2013.

[5] Q.V. Le. Building high-level features using large scale unsupervised learning, in: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2013.

[6] R. Jhawar, V. Piuri, M. Santambrogio, Fault tolerance management in cloud computing: a system-level perspective, IEEE Systems Journal 7(2)(2013) 288-97.

[7] G.L. Rosen, E.R. Reichenberger, A.M. Rosenfeld, NBC: the naive bayes classification tool web server for taxonomic

classification of meta genomic reads, Bioinformatics 27(1)(2011) 127-9.

[8] S. Landset, T.M. Khoshgoftaar, A.N. Richter, T. Hasanin, A survey of open source tools for machine learning with big data in the Hadoop ecosystem, Journal of Big Data 2(1)(2015) 1-36

[9] I.H. Witten, E. Frank, M.A. Hall, C.J. Pal, Data Mining: Practical Machine Learning Tools and Techniques, second ed., Morgan Kaufmann, San Francisco, 2016.

[10] J. Dean, S. Ghemawat, MapReduce: a flexible data processing tool, Communications of the ACM 53(1)(2010) 72-7.

[11] A. Holmes, Hadoop in Practice, Manning Publications, Shelter Island, 2012.

[12] Nagios core version 3.x documentation. <http://nagios.sourceforge.net/docs/nagioscore-3-en.pdf/>, 2010 (accessed 02.02.10).

[13] Rackspace private cloud powered by OpenStack. <https://www.rackspace.com/cloud/private/openstack solutions/openstack />, 2017 (accessed 12.03.17)

[14] Aneka: enabling. NET-based enterprise grid and cloud computing. <http://www.manjrasoft.com/ products.html/>, 2017 (accessed 07.03.17).

[15] R.R. Bouckaert, E. Frank, M.A. Hall, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten WEKAâ˚’experiences with a Java open-source project, Journal of Machine Learning Research 11(9)(2010) 2533-2541.

[16] N. Saswade, V. Bharadi, Y. Zanzane, Virtual machine monitoring in cloud computing, Procedia Computer Science 1(79)(2016) 135-142.

[17] G. Aceto, A. Botta, W. De Donato, A. Pescapè, Cloud monitoring: a survey, Computer Networks 57(9)(2013) 2093-2115.

[18] A. Kumar, R. Shankar, A. Choudhary, L.S. Thakur, A big data MapReduce framework for fault diagnosis in cloud-based manufacturing, International Journal of Production Research 54(23)(2016) 7060-7073.

[19] W.E. Smith, K.S. Trivedi, L.A. Tomek, J. Ackaret, Availability analysis of blade server systems, IBM Systems 47(4)(2008) 621-640.

[20] A. Alkasem, H. Liu, Z. Decheng, Y. Zhao, AFDI: A virtualization-based accelerated fault diagnosis innovation for high availability computing, arXiv preprint arXiv 1507.08036, 2015.

[21] B.E. Guo, H.T. Liu, C. Geng, Study on hybrid-weight for feature attribute in naïve bayesian classifier, In: Proc. 2014 Fifth International Conference on Intelligent Systems Design and Engineering Applications (ISDEA), 2014.

[22] A. Gupta., Learning Apache Mahout Classification, Packt Publishing, Birmingham, 2015.

[23] B.P. Sharma, P. Jayachandran, A. Verma, C.R. Das CloudPD: problem determination and diagnosis in shared dynamic clouds, in: Proc. 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2013.

[24] L. Wang, G. Von Laszewski, A. Younge, X. He, M. Kunze, J. Tao, C. Fu, Cloud computing: a perspective study, New Generation Computing 28(2)(2010) 137-46.

[25] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares, X. Qin, Improving mapreduce performance through data placement in heterogeneous hadoop clusters, in: Proc. 2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010.

[26] A. Eldawy, M.F. Mokbel, Spatialhadoop: a mapreduce framework for spatial data, in: 2015 IEEE 31st International Conference on Data Engineering (ICDE), 2015.

[27] S. Haloi, Apache ZooKeeper Essentials, Packt Publishing, Birmingham, 2015.

[28] S.B. Kim, K.S. Han, H.C. Rim, S.H. Myaeng, Some effective techniques for naive bayes text classification, Knowledge and Data Engineering, IEEE Transactions 18(11)(2006) 1457-66.

[29] C.W. Tsai, C.F. Lai, H.C. Chao, A.V. Vasilakos, Big data analytics: a survey, Journal of Big Data 2(1)(2015) 21.

[30] V.R. Eluri, M. Ramesh, A.S.M. Al-Jabri, M. Jane, A comparative study of various clustering techniques on big data sets using apache mahout, in: Proc. 2016 3rd MEC International Conference on Big Data and Smart City (ICBDSC), 2016.

[31] A. Bechini, F. Marcelloni, A. Segatori, A MapReduce solution for associative classification of big data, Information Sciences (332)(2016) 33-55.

[32] G.E. Box, G.M. Jenkins, G.C. Reinsel, G.M. Ljung, Time Series Analysis: Forecasting and Control, John Wiley & Sons, New York, 2015.

[33] V. Krishnan, Probability and Random Processes, John Wiley & Sons, New York, 2015.

[34] X. Wang, X. Sun, An improved weighted naive bayesian classification algorithm based on multivariable linear regression model, in: Proc. 2016 9th International Symposium on Computational Intelligence and Design (ISCID), 2016.

[35] L. Zhou, H. Fujita, Posterior probability based ensemble strategy using optimizing decision directed acyclic graph for multi-class classification, Information Sciences (400)(2017) 142-156.

[36] A. Alkasem, H. Liu, D. Zuo, Utility cloud: a novel approach for diagnosis and self-healing based on the uncertainty in snomalous metrics, in Proc. the 2017 International Conference on Management Engineering, Software Engineering and Service Sciences (ICMSS '17), 2017.

[37] P.J. Chuang, C.Y. Chou, Efficient concurrent virtual machine scheduling for Xen hypervisors, in: Proc. Computer Science, Technology and Application: Proceedings of the 2016 International Conference on Computer Science, Technology and Application (CSTA2016), 2016.

[38] A. Babu, M.J. Hareesh, J.P. Martin, S. Cherian, Y. Sastri, System performance evaluation of para virtualization, container virtualization, and full virtualization using xen, openvz, and xen server, in: Proc. 2014 Fourth International Conference on Advances in Computing and Communications (ICACC), 2014.

[39] D. Seward, Applied Statistics in Business and Economics, McGraw-Hill Education, New York, 2009.

[40] L. Lebart, Correspondence analysis, in: Proc. Data Science, Classification, and Related Methods: Proceedings of the Fifth Conference of the International Federation of Classification Societies (IFCS-96), 2013.

[41] M. Kateri, Contingency Table Analysis, Springer, New York, 2014.

[42] E. Gawrilow, S. Hampe, M. Joswig, The polymake XML file format, in: Proc. International Congress on Mathematical Software, 2016.

[43] F.H. Tseng, L.D. Chou, H.P. Chiang, W.J. Yu, Implement efficient data integrity for cloud distributed file system using merkle hash tree, Journal of Internet Technology (15)(2014) 307-316.

[44] C.A. Mertler, R.V. Reinhart, Advanced and Multivariate Statistical Methods: Practical Application and Interpretation, Routledge, London, 2016.