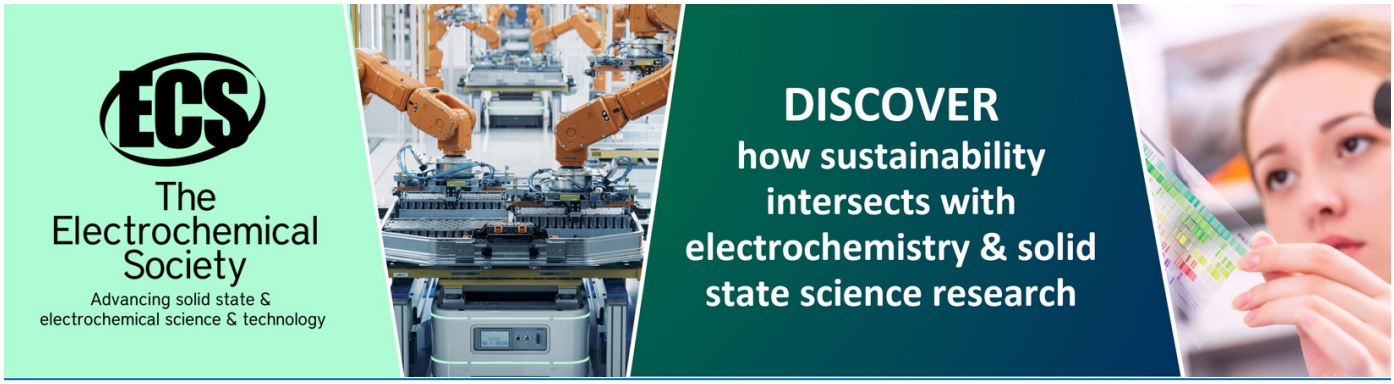
Cloud Computing: A model Construct of Real-Time Monitoring for Big Dataset Analytics Using Apache Spark



Cloud Computing: A model Construct of Real-Time Monitoring for Big Dataset Analytics Using Apache Spark

Ameen Alkasem[1], Hongwei Liu[1], Decheng Zuo[1] and Basheer Algarash[1]

[1]School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China

Email: 77ameen@gmail.con

Abstract. The volume of data being collected, analyzed, and stored has exploded in recent years, in particular in relation to the activity on the cloud computing. While large-scale data processing, analysis, storage, and platform model such as cloud computing were previously and currently are increasingly. Today, the major challenge is it address how to monitor and control these massive amounts of data and perform analysis in real-time at scale. The traditional methods and model systems are unable to cope with these quantities of data in real-time. Here we present a new methodology for constructing a model for optimizing the performance of real-time monitoring of big datasets, which includes a machine learning algorithms and Apache Spark Streaming to accomplish fine-grained fault diagnosis and repair of big dataset. As a case study, we use the failure of Virtual Machines (VMs) to start-up. The methodology proposition ensures that the most sensible action is carried out during the procedure of fine-grained monitoring and generates the highest efficacy and cost-saving fault repair through three construction control steps: (I) data collection; (II) analysis engine and (III) decision engine. We found that running this novel methodology can save a considerate amount of time compared to the Hadoop model, without sacrificing the classification accuracy or optimization of performance. The accuracy of the proposed method (92.13%) is an improvement on traditional approaches.

1. Introduction

Large datasets monitored move fast in real-time and tend to be the most valuable as a result [1]. For example, utility cloud service providers may wish to monitor engines to detect faults or anomalies in seconds or quickly engage self-recovery before losing service. To enable these low-latency processing applications, there is a need to design and evaluate a new model to rapidly manage and analyze huge data by means of streaming computation models that scale transparently to large clusters such as Apache Spark [2]. This happens with cheaper applications and low maintenance costs. Among the recent tools and technologies, Apache Spark has become one of the most popular engines for large dataset processing. Evaluating and designing a new model is challenging. Monitoring is an important aspect of systems engineering allowing effective maintenance and evaluation of deployed systems [3]. There is a common set of motivations for monitoring which apply to virtually all areas of computing, including cloud computing; perhaps foremost is capacity planning failure or underperformance

detection, redundancy detection, system evaluation and policy violation detection. At their very simplest, monitoring and control entail a three stage process illustrated in Figure 1: (I) The collection of relevant state (II) The analysis of the aggregated state and (III) Decision making as a result of the analysis [4].

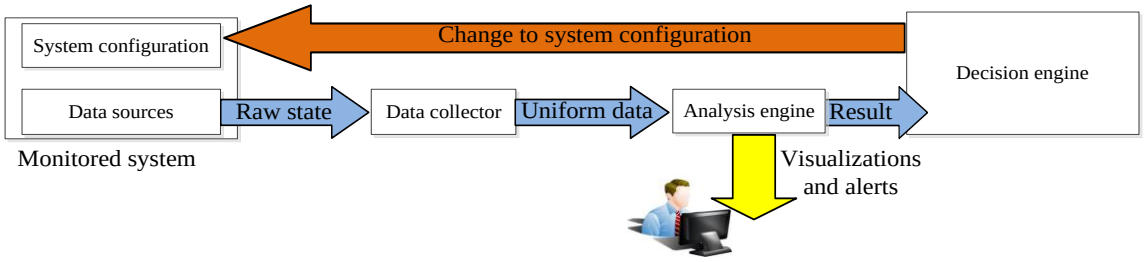


Figure 1. A three stages monitoring process technical

More precisely, professionals use machine learning technologies such as Bayesian Networks (BNs) extensively for the classification and analysis of datasets due to their "learning" capabilities from the combination of datasets. These are often based on training and tests in order to make relatively highly accurate real-time decisions. Apache Spark is a fast general purpose computing engine designed for large-scale data processing. Spark has the advantages of Hadoop of MapReduce; Spark is different from MapReduce job intermediate results can be stored in the memory, thereby eliminating the need to read and write Hadoop Distributed File System (HDFS), so Spark can be better applied to the big dataset and machine learning iterative MapReduce algorithm. Recently, research in this area has abounded. Zhang, et al. [5] provided a flexible architecture called *CloudMonatt* to monitor the security health of customers' VMs within a cloud system. Le et al. [6] developed an efficient multilevel healthy cloud system to analyze data collected from CGU smart clothes using Spark. Aceto et al. [7] surveyed cloud monitoring wherein they identified open issues, main challenges, and future directions in the field. Kumar et al. [8] developed a MapReduce framework for automatic pattern recognition based on fault diagnosis by solving data imbalance problems in cloud-based manufacturing (CBM). Melo et al. [9] developed and proposed a model to evaluate nodes' capacity in a cloud computing environment based on available hardware resources. Nevertheless, comprehensive analysis yields the fact that these researchers, who work in different research areas, did not take into account other effects, such as efficiency, accuracy, and optimization of performance in real-time monitoring of large datasets analysis and classifiers as they pertain to diagnosing faults and anomalies' behaviors.

The contribution of this paper presents a new methodology which includes machine learning algorithms and the Apache Spark to accomplish fine-grained monitoring and control for the fault diagnosis and recovery of the IaaS clouds in real-time. This paper presents a new methodology for constructing a model that optimizes performance of real-time monitoring and improves prediction accuracy. Also, this research performs scalability monitoring to see the speedup of the big datasets processing and analyze this in real-time based on Spark Streaming. The case study here is the failure of virtual machines (VMs) to start up. The methodology proposition ensures that the most sensible action is carried out during the procedure of fine-grained monitoring and generates the highest efficacy and cost-saving fault diagnosis through three construction control steps: (I) Data collection; (II) Analysis engine; and finally (III) Decision engine, as shown in Figure 2. Results show that running the

new model across machine learning algorithms and Apache Spark can save a consider able amount of time compared to running the model against a Hadoop without sacrificing the classification accuracy. It can also optimize analytical performance. Results have proven promising, considering an accuracy of 92.13% with an increase in the size of the datasets.

This paper has five sections, beginning with this introduction. In section 2, we describe the tools we used in an empirical evaluation of our methodology. Section 3 focuses on our approach's materials and methods, and explains the analysis, classifier and decision engine models. Section 4 explicates the experimental results, environment and includes a discussion. Finally, we provide our conclusions and future works in Section 5.

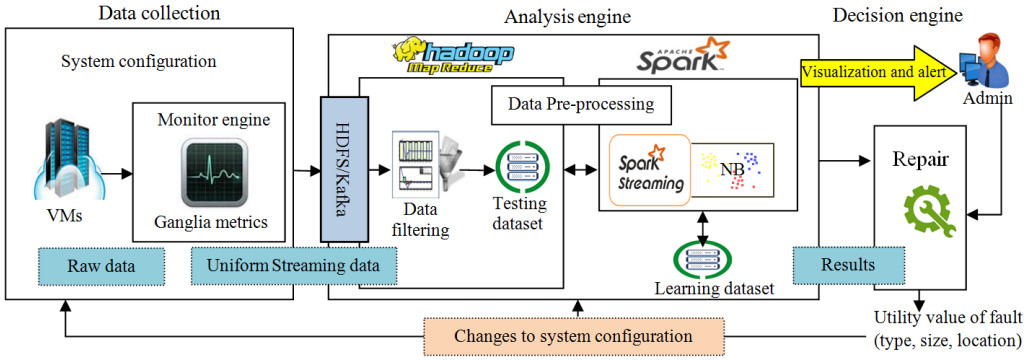


Figure 2. Overview of a new model monitoring workflow based on Apache Spark

2. Tools Used in Empirical Evaluation of Methodology

2.1 Apache Spark Technology

Spark is a distributed computing framework like MapReduce, but it score is a flexible distributed data set that provides a richer model than MapReduce. Specifically, it allows iterations of data sets in memory quickly to support complex data mining algorithms and graph calculation algorithms [3]. It provides a simple programming interface (API's) for various analytics algorithms including real-time data streaming, SQL Queries, graph processing and machine learning. The interface allows for an application developer to easily use the CPU, memory, and storage resources across a cluster of servers for processing large and complex datasets. The advantages of Spark Streaming are:(I) runs on 100+ nodes, and reaches the second delay, (II) memory-based execution engine, with efficient and fault-tolerant features, (III) integration of Spark's batch and interactive queries, and (IV) Up to 10 times faster than Hadoop MapReduce.

Apache Spark is a very simple architecture with only two nodes (Master and Worker) that run with a cluster manager such as Spark, Hadoop or others, as shown in Figure 3

Figure 3. Apache Spark architecture

The Spark program processes the batch by manipulating the interfaces that RDDs (Resilient Distributed Datasets) provide, such as map, reduce, filter and so on. In Spark Streaming, the interfaces provided by the operation of the DStream (RDD sequence representing the data flow) are similar to those the RDD provides.

2.2 Spark Streaming Framework

Spark Streaming is a real-time native computing framework that extends Spark's ability to handle large-scale streaming data. The framework demands data collection from many resources such as TCP sockets, HDFS and Kafka, as well as processing it through complex algorithms expressed with high-level functions such as MapReduce and window [13]. The basic principle of Spark Streaming is to split the input data stream into units of time slices (seconds), and then process each time slice data in a batch-like manner. Figure 4 illustrates the basic principle.

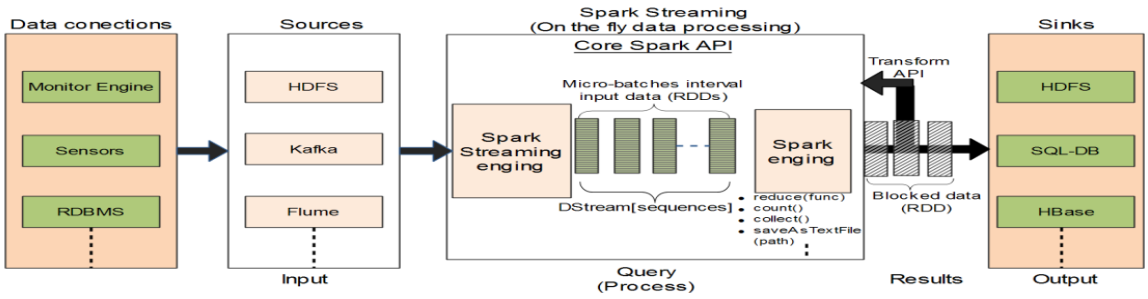


Figure 4. Spark Streaming implementation framework

We used Spark Streaming in this work because it allows us to combine applications such as streaming, batch and interactive queries through rich APIs and memory-based high-speed computing engines. Indeed, it is suitable for applications that require historical data and real-time data combination analysis, though real-time requirements do not require particularly intensive application. In addition, the RDD data reuse mechanism can be more efficient and more fault-tolerant processing [13].

2.3 Structured Streaming

Structured stream processing is another engine based on the Spark SQL optimization engine Catalyst optimizer, ensuring its performance is also strong. This kind of processing ensures data is processed only once, while achieving end-to-end data and high fault tolerance by investigating mechanisms like checkpoints and write ahead logs. The current dataset/DataFrame API supports program languages in Scala, Java, and Python. We can implement flow aggregation, event window, join aggregation, and so on [13].

2.4 Machine Learning with Apache Spark

Machine learning is a technology with strong ties to statistics and optimization; these allow learning from existing data to explore hidden valuable information [14]. It has become one of the most popular techniques for knowledge discovery and predictive analytics, especially with the current exponentially growing data derived from various disciplines such as the medical sciences and business. Many

applications, like spam filtering, advertisement targeting, computer vision, and bioinformatics, have adopted machine learning algorithms to better guide leadership's decisions. Apache Spark programming model and the Hadoop platform to analytic and process big datasets in real-time gives a powerful algorithms implementation. Many machine learning algorithms have been investigated to be transformed to the Spark paradigm in order to make use of the HDFS/Kafka resources [15]. The Na ïve Bayes classifier (NBC) is one of the supervised learning classification algorithms one can program along the lines of Spark. Currently, the NBC is a scalable machine learning library that supports large dataset processing [16]. Moreover, one can train the NBC very efficiently. Within a single training pass, it computes the conditional probability distribution of each feature given label and then applies Bayes' Equation (1) to compute the conditional probability distribution of a label, given an observation, then employs it for prediction [17].

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$
$$P(c|X) = P(x_1|c) \times P(x_2|c) \times \ldots \times P(x_n|c) \times P(c) \quad (1)$$

where

$P(c|x)$: is the posterior probability of the class (target) given predictor (attribute).
$P(c)$: is the class prior probability.
$P(x|c)$: is the likelihood probability.
$P(x)$: is the predictor prior probability (evidence).

3. Our Approach

A new methodology model used the advantages NBC models and Apache Spark Streaming technical. This provided a new approach and framework of monitoring and control of a massive amount of datasets and an analysis in real-time at scale. The new model reduces the cost of the time of a fault diagnosis through fast analytic and engages in classifying large testing and training datasets based on the Spark Streaming technical and HDFS or Kafka storage platforms. The new model monitors a wide range of metrics across the VMs as shown in Figure 5. Most prior work monitored metrics based on the risk levels of consequences according to the symptoms [18].

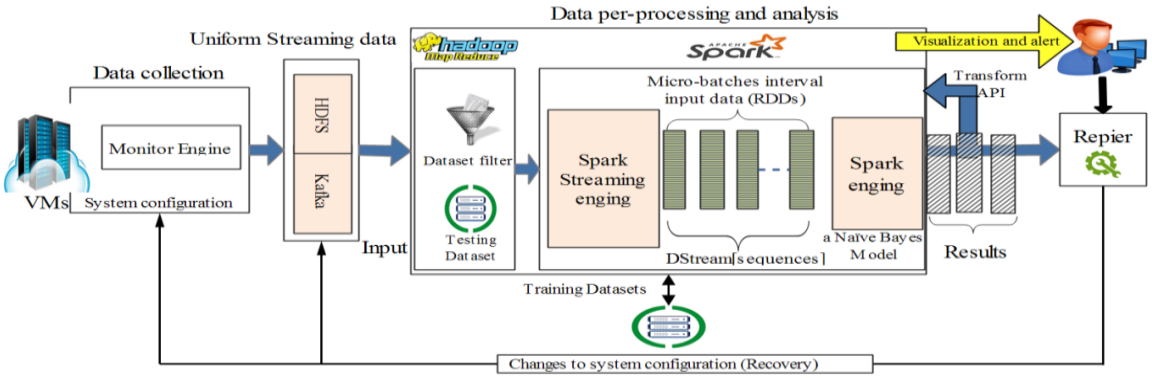


Figure 5. A new methodology proposed framework based on Spark

3.1. The Topology of a New Methodology Proposed

3.1.1. Data Collections for Proposed Model

In this paper, we collected data from large monitoring engines to obtain the metrics of interest (numerical predictors or attributes). Then, we collected the metrics of the VMs by running the VMs on the Xen-hypervisor [19] installed on the host server in combination with pre-processing (reported in Table 1) using Ganglia metrics software [20], as shown in Table 2. Because these metrics exist in the form of numerical data, the numerical variables had to be transformed into their categorical counterparts (binning) before constructing their frequency table to use them as input for the proposed model topology structure. Therefore, as shown in Figure 6, the pre-processing step consists of four steps to translate continuous percentage utilization into interval probability values and generate monitoring vectors of events (M-events) by using our method [21]. We adopted this method with a filtering dataset to remove and process the outlier data and noise using the Extended Kalman Filter (EKF) [22]. As a result, we generated a new method algorithm for transformed a numerical data to binning data (Figure 7).

Table 1. Fault reasons for the start-up failure of VM/Host metrics

Fault category	Fault causes	Measurement Level
CPU utilization	% CPU time used by host CPU during normal sampling period (Host CPU usage) % CPU time during which the CPU of the VM was actively using the physical CPU (VM CPU usage) % CPU time during which the CPU of the VM was ready but could not get scheduled to run on the physical CPU (VM CPU Ready-Time)	VM, Host
Memory usage	% of used memory (Memory usage)	VM, Host
Network overhead	% of network usage (Bandwidth)	VM, Host
I/O storage usage	% of disk usage (Throughput)	Host

Table 2. A sample data collection of CPU-utilization (testing dataset)

Time- Monitoring	VM-CPU-Ready-Time	Host-CPU-usage	VM-CPU-usage
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
⋮	⋮	⋮	⋮

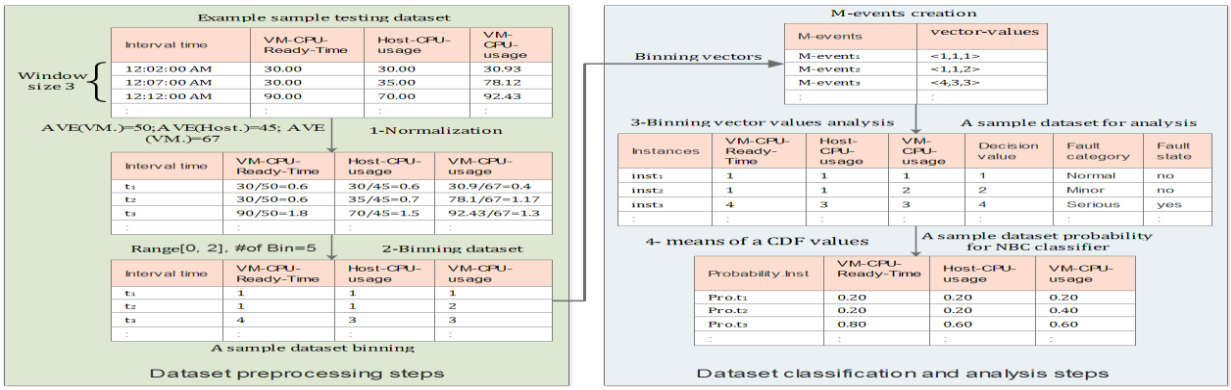


Figure 6. A process method steps and a sample of tables of the datasets transformed

Algorithm 1: Transformed and filtered a data points to data binning

```
Input: Metrics_values[i, j], mean[j], m, r, n, num.
Output: Data binning table.
1. For i=0 , i<=n; i++ do
2.     For j=0,j<=num, j++ do
3.        Normalized[i,j]=Metrices_values[i,j]/mean[i]
4.          If Normalized [i,j]>n then
5.              Bin[i,j]=m,
6.          else
7.             Bin[i,j]=TRANC(Normlized[i,j]/(r/m)),
8.     End for
9.  End for
10. Data_normalized[i,j]=Filtering-dataset(Bin[i,j]),
11. End Function return(Data_binning).
```

Figure 7. A proposed method algorithms for filtering and transformed a numerical data to data binning

Our method has a buffer size of *n* (a look-back window) for the metrics of the previously observed n samples (e.g., *n*=3, range [0, 2] and #Bin=5). The look-back window is used for multiple reasons: (I) shifts of work patterns may render old history data even useless or misleading, (II) at exascale, it is impractical to maintain all history data, and (III) it can be implemented in high speed RAM, which can further increase detection performance. We use a window-size of 3, sampling interval of 5 seconds, and interval length of 14 minutes. The number of data points in the 14 minutes interval are, thus, (14*60)/(3*5) = 56. The metrics in the look-back window at each time instance (t_1,t_2,..., t_i, i is number of instances) serve as inputs for the pre-process, as shown in Table 3, so that m-events (M_1, M_2,..., M_i) creation are the input for every component metric for our presented methodology, as shown in Table 4. The binning value and the decision value are determined by the following formulas, (2), (3) and (4):

$$\begin{aligned} &\textbf{If}(x > 2)\ \textbf{then}\ \textit{binning value} = 5, \\ &\textbf{else}\ \textit{binning value} = \ \text{TRUNC}(x/0.4) \end{aligned} \tag{2}$$

$$\text{Decision } \textit{value} = \text{MAX}(\textit{binning value}) \tag{3}$$

Where x is the normalization value for the attributes and 0.4 is a statistic suggested by the probability values. After this, we can begin the classification with a dataset probability (predictor) by means of a cumulative distribution function (CDF) [23] as Table 5 illustrates. For a continuous random variable, the CDF equation is.

$$P(X \leq x) = \frac{x - a}{b - a} \tag{4}$$

Where a is the lower limit 0 and b is the upper limit 5, $a \leq x \leq b$.

Table 3. A sample of data normalization used for binning

	Sample	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
	⋮	⋮	⋮	⋮

Table 4. A sample of dataset binning with decision values

M-events	CPU	Network	Memory	Decision value
M_1	1	1	1	1

Window size 3	M_2	1	1	2	2
	M_3	4	3	3	4
	M_4	2	2	2	2
	⋮	⋮	⋮	⋮	⋮

Table 5. A sample of the classifier probability (predictor) dataset before training

Process Instance	CPU	Network	Memory	VM/Host state	Fault state
$Pro.inst_1$	0.20	0.20	0.20	Normal	**?**
$Pro.inst_2$	0.20	0.20	0.40	Minor	**?**
$Pro.inst_3$	0.80	0.60	0.60	Serious	**?**
$Pro.inst_4$	0.40	0.40	0.60	Minor	**?**
⋮	⋮	⋮	⋮	⋮	⋮

3.1.2. Analysis Engine Model

We present a new methodology for large-dataset analysis based on the Hadoop MapReduce and Apache Spark platform, and we implement new algorithms with MapReduce and Spark Streaming to achieve high-performance and efficient classification for analyzing collections of large metrics of test and learn datasets in real time. This section explains the new algorithms and workflow of analysis, as well as the pre-processing method. Figure 8 elucidates the procedure by displaying four new modules to Hadoop along with the techniques we used for method evaluation (Apache Spark):

(1) A monitoring engine: Monitors each VM and host server to collect the metrics of interest (numerical predictors or attributes).
(2) Pre-processing: we test new data from the metrics collection monitor by normalizing it and binning it into intervals, then using the raw time-series data to generate monitoring events (m-events).
(3) Diagnosis engine: Our model analyzes the time series' entropy to locate patterns that signify faults or anomalies in the system being monitoring.
(4) A decision process model: In this model, a look-back window with size 3 creates a dataset table according to the following decision values:

- **If** (0 =decision value < 2) **Then** (fault category is "Normal," node fault state is working "fault=no").
- **If** (2 <decision value =3) **Then** (fault category is "Minor," node fault state is working "fault=no").
- **If** (decision value > 3) **Then** (fault category is "Serious," node fault state is no working "fault=yes").

Our work focuses on enhancing the new proposed methodology model for differing sizes of datasets and assessing the Hadoop MapReduce program and Apache Spark Streaming in having the capability of learning from a past large datasets with the purpose of achieving high-performance fault diagnosis for the metrics of the VMs and host server collected in real time monitoring. Figure 8 demonstrates how we can take raw data from the testing metric data collected by large monitoring engines (Xen-Hypervisor) as shown in Table 9 in Appendix A, and a historical training dataset as shown in Table 10 in Appendix A.

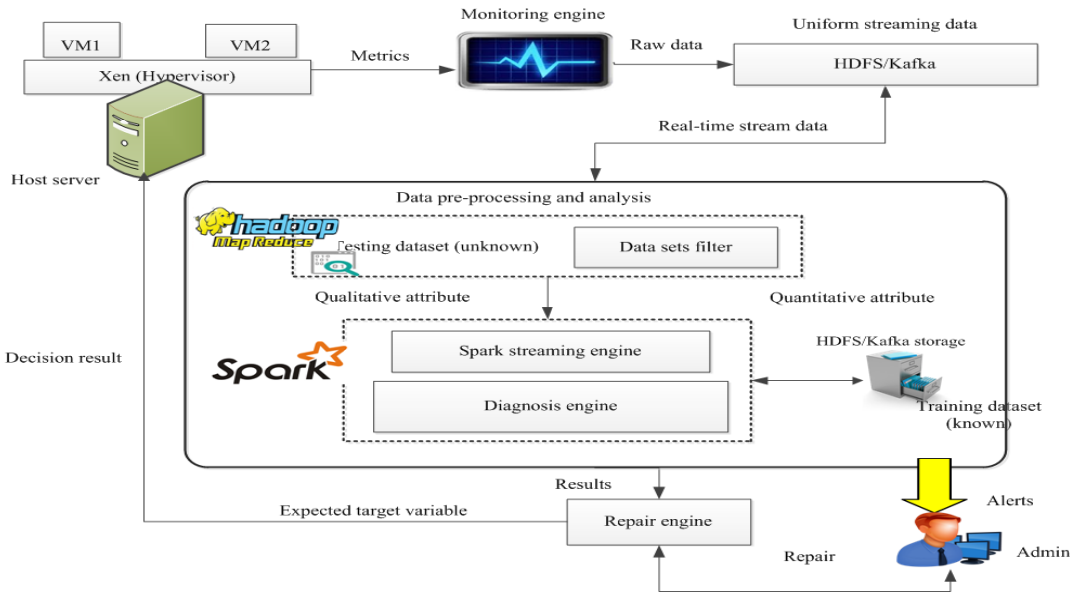


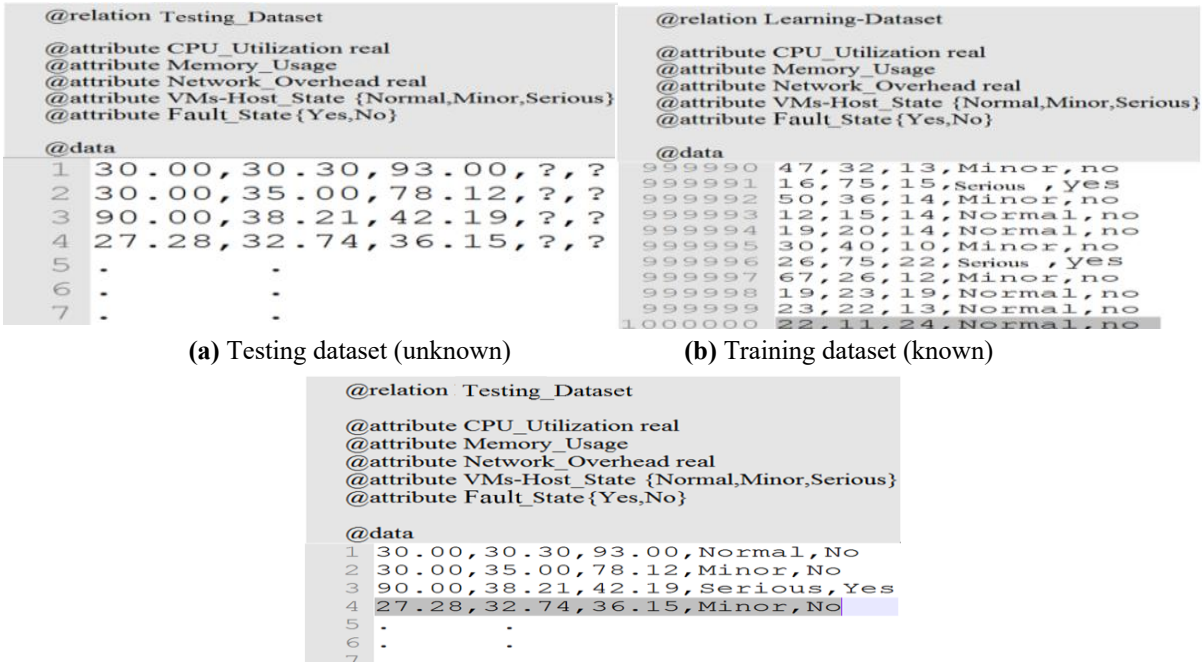
Figure 8. Proposed frameworks for a fine-grained approach with Spark

3.1.3. Diagnosis and Classifier Engine

The proposed diagnosis approach is a general concept representing a combination of analytical tools that all cooperate to form the analysis and diagnosis model (Figure 8). We took our raw data from large engines monitoring a host server, as well as drawing from data research communities have already collected. A monitoring engine collected and processed measurement data such as basic resource metrics for each VM and host server, as listed in Table 1. The diagnosis engine contains the storage of Kafka/HDFS and hybrid intelligent models Hadoop MapReduce and Spark. An HDFS/Kafka is a storage platform for a real-time streaming data collected. Here, we can use the Hadoop to quickly obtain the overall generating testing datasets and input them in the pre-processing and analysis model.

The Spark streaming analyzes m-events diagnosis by pre-processing and the use of I/O pairs to calculate the parameters estimated by an NBC in the Spark engine for large learning datasets. We utilized three statistical measures (recall, precision and accuracy) to evaluate the effectiveness of fault diagnosis in real-time using the Spark streaming engine in Hadoop for large-testing/learning dataset problems; Subsection 3.1.4 and Section 4 unveils more details and results. The data parser first pre-processes all component metrics recorded in a common XML file [24]. Each file contains two parts after pre-processing: (I) The attributes of the components and (II) The dataset, as shown in Figure 9. The model records each observation of component metrics as a single line of datasets 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, no). A comma separates each column, which is useful because, by default, Hadoop MapReduce splits the input files by line and passes each line to a mapper function. It then stores all preprocessed component metrics in the master node as a repository while waiting for further sampling. The proposed approach maintains a buffer size of 3 (a look-back window) of the last *n* samples' metrics.

The metrics observed in the look-back window at each time instance serve as inputs to be pre-processed.



(a) Testing dataset (unknown) **(b)** Training dataset (known)

(c) Testing dataset after classification and learning (known)

Figure 9. Datasets modeling (XML format)

3.1.4. Decision Engine Model

The classification of component states and system states is a key step in workflow. Figure 8 shows the job sequence of this step. When the real-time test dataset is ready in HDFS/Kafka, a new model starts the training job to build a model. The algorithm combines the test data with the model and generates an intermediate table, then classifies the job and simultaneously computes the probability of each component in the three classes. Finally, it makes a decision about the final system state and records statistics regarding the contingency table and intermediate values. In our experiments, we used two datasets: (I) New testing datasets (unknown) and, (II) Training datasets (known). Tables 6 and 7 show how these data sets work by converting to XML format for input to the proposed model algorithms. An example of the XML files formation (Figure 9(a)-(b)) is our simple dataset framework for a state-based probability model that predicts component utilization. In Figure 10, the measurement level varies from 0 to 100%. The component utilization uses CPU ϵ {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 by combining the test and training datasets (Figure 9(c)). In the process, we first define our task to classify the three state components using a model proposed. 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 utilization, memory usage, and network represent three basic components while host state server represents the system state (Figure 10). This

results in only three classes (normal, minor and serious) for the component state measurements and only two (yes, no) for the system fault state [25]. To simplify the problem, we choose the same number of normal, minor, and serious measures for the components. We must then convert the classification problem to a counting problem on the training datasets and test datasets.

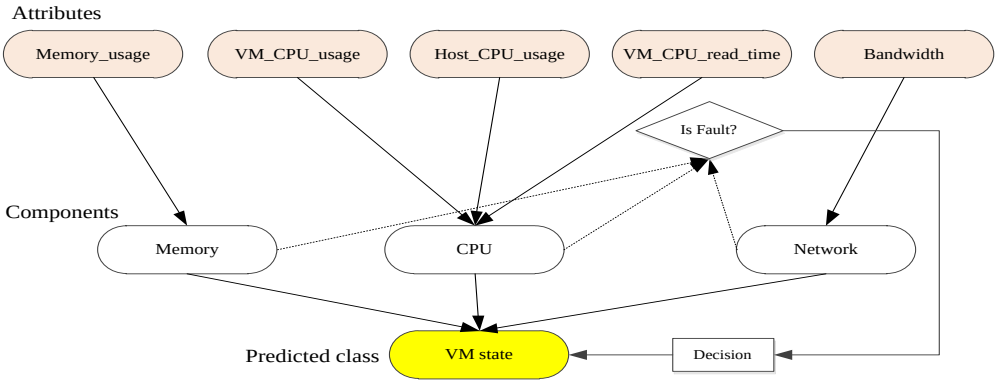


Figure 10. Model predicting component utilization for VM

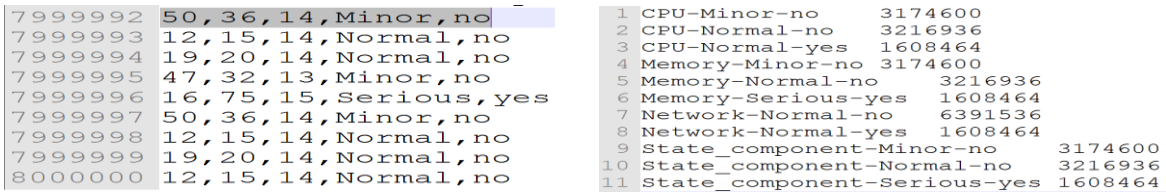
We divided the problem into two procedure programs:

1) The first is a combination of two java and Scala programs for monitor and control of big dataset training, filtering, and streaming datasets (Figure 16 in Appendix B). All training and testing instances are fed into these programs to produce a model for all unique attributes and filtering datasets with their frequencies of normal, minor and serious component states. The model also contains information necessary for the final classification in real-time processes based on Apache streaming engine.
2) The second program regards classifying and analyzing the test and training dataset, as show in Figure 17 in Appendix B. In this program, the training and the testing dataset measures combine into an intermediate table with all information necessary for the final classification. This program classifies all instances for the test dataset and gets the final results of fault state classes (yes, no). Moreover, it writes final resulting classification to HDFS or Kafka storage platforms.

After the two programs finish, the results collector retrieves the model classification results intermediate values table and test data statistics from HDFS/Kafka storage. By the end of these programs, instances have been classified and evaluation into a yes/no final fault state with normal, minor and serious state classes, which one can view in Table 6 and Figure 11.

Table 6. A sample of the dataset classifier probability after training

Process. Instance	CPU	Network	Memory	VMs/Host state	Fault state
$Process.inst_1$	0.25	0.25	0.25	**Normal**	**no**
$Process.inst_2$	0.25	0.25	0.50	**Normal**	**no**
$Process.inst_3$	1.00	0.75	0.75	**Serious**	**yes**
$Process.inst_4$	0.50	0.50	0.50	**Minor**	**no**
$Process.inst_5$	0.40	0.31	0.60	**Minor**	**no**
⋮	⋮	⋮	⋮	⋮	⋮



(a) Testing dataset after diagnosis and analysis **(b)** Spark streaming result for big dataset testing

Figure 11. Implementation results of proposed approach with new programs algorithms

4. Experimental Results

4.1. Experiment Environment

For experimental purposes, we implemented our proposed model's programs using Scale and Java coding. All programs ran on several Apache Spark clusters with different number of nodes. The experiment setup used 2 VMs (VM1 and VM2) on a Xen-Hypervisor platform hosted on one Dell blade server. Apache Spark cluster formed through a local server machine service. Each node inside the cluster has the following configuration: vCPU (3.9 GHz), 1 TB storage and 64 GB RAM. The dataset contains multi instances stored inside local server machines by HDFS/Kafka storage management then read through the master node. The architecture of the Apache Spark cluster appears in Figure 12. For comparison, we have also implemented the same environment using Hadoop platform without Spark.

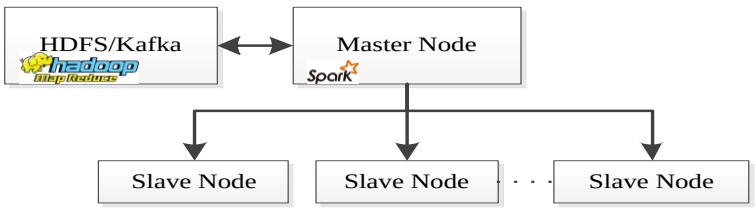
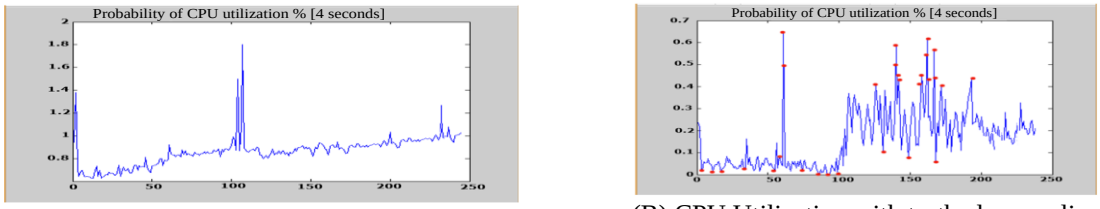


Figure 12. Architecture of the experiment of Apache Spark's cluster

4.2. Evaluation with Real-time Monitoring

We implemented our approach at both the VMs level and the host server level, so the local time series is calculated first for the VMs. Their aggregation for the host server's global time series follows. The first implementation uses Xen-Hypervisor and the Ganglia metrics method to record and identify global resource provisioning anomalies [26]. We injected forty anomaly samples into the testbed, leading to global resource consumption by the anomalies/faults without excluding the CPU-utilization of the running host server (Figure 13). We collected the VM and host metrics using the Ganglia metrics method and analyzed them in a fault detector and classifier. To test the scalability and performance of the new model, we varied the size of the datasets size from 8,000,000 to more than 64,000,000 instances of recorded metrics in each class with datasets between 1.26 GB and 12 GB or over.



(A) CPU Utilization availability (B) CPU Utilization with testbed anomalies

Figure 13. CPU Parameters using Ganglia monitored metrics

4.3. Results

The resulting statistics include the classification frequency and accuracy of CPU utilizations for two VMs and host server machines by programs algorithms (Figure 14). These results demonstrate a high accuracy for the classification of component states (normal, minor and serious). We show the accuracy and throughput of the system evaluation by Hadoop and Apache Spark in Figure 15.

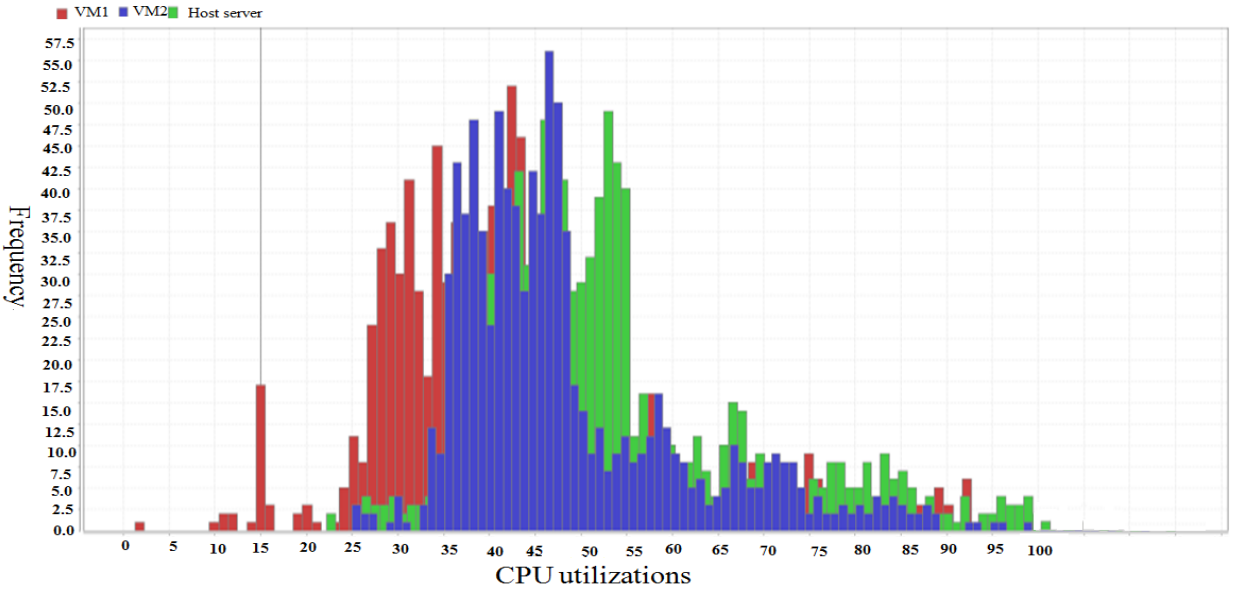


Figure 14. Histogram frequency of CPU utilizations running on VM1, VM2 and Host server

4.4. Discussion

Equations in Table 7's four statistical measures [27][28] helped us evaluate the effectiveness of the construction steps of the monitor procedure, thereby allowing us to detect the faults in the testbed of our experimental setup. We explain this further in Section 4.1. Our experimental results reveal several interesting findings of the evaluation of NBC in our model with Apache Spar. We also committed to a performance summary and results of different implementations without the use of Spark (Figure 15). We achieved high accuracy of up to 92.13% and a 3–15% false alarm rate. The use of Apache Spark Streaming can be optimized to speed up the parameter learning stage in classification and analysis for a wide range of input and dataset sizes even very large ones. The number of instances the system can process and analyze in one second increased from 1.20 GB to more than 12 GB. Table 8 presents a performance summary and various results.

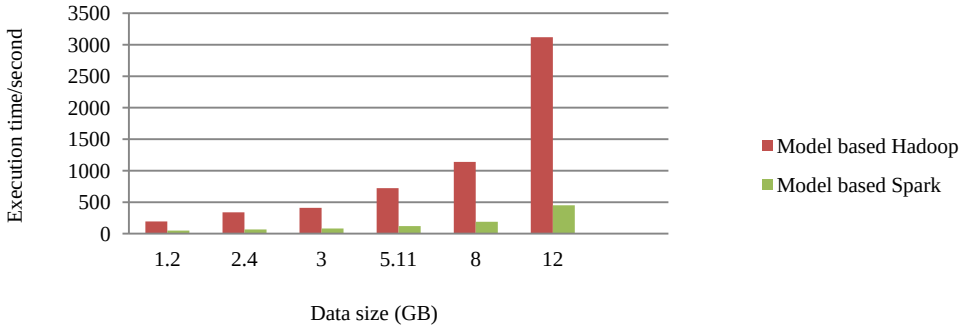


Figure 15. Throughput of system of different model implementations based on Hadoop and Spark

Table 7. Four measures in statistics

Precision	Recall	Accuracy	False-alarm rate (FAR)
$\frac{successfuldetections}{oftotalalarms}$	$\frac{successfuldetections}{oftotalanomalies}$	$\frac{2 * precision * recall}{precision + recall}$	$1 - presision$

Table 8. Performance summary results

Model	Accuracy (%)	Time(Second)*10^3	Dataset size(GB)
Proposed Model with Hadoop	72.86	3.210	12
A new Proposed Model with Apache Spark	**92.13**	**0.450**	12

Finally, our new approach methodology using Apache Streaming can scale over 64 million instances of recorded metrics in each class from a monitoring engine with a high accuracy and saving cost of the time compared to other models.

5. Conclusion

In this paper, we presented a new fine-grained fault-tolerance platform for monitoring and control methodology of big datasets in real-time based on Spark Streaming. Our methodology proposition ensures that the most sensible action occurs during the procedure of fine-grained monitoring, generating the highest possible efficacy and cost-saving fault diagnosis through three construction control steps: (I) data collection, (II) an analysis engine, and (III) a decision engine via Apache Spark. The additional modules inserted to evaluate the newly proposed algorithms resulted in good classification with a high accuracy of up to 92.13% with a 3–15% false alarm rate. Based on the experimental results, a high-performance evaluation of real-time monitoring and control for large datasets using Apache Spark has a better average running time than when using Hadoop. When the size of big datasets is increases, the results on the Apache Spark cluster has good scalability and has better-than-average running time. The key ideas in future work, we have planning to extend our work to develop and design a new model of performance testing for shared dynamic cloud services. The model will be achieves end-to-end performance testing management framework that can troubleshoot, analysis, classify and recovery actions for virtualized cloud based fault bottlenecks and anomalies behavior.

Appendix A. A sample datasets

Table 9. A sample dataset of testing metrics before training and classification

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	**?**	**?**
12:27:00 AM	32.1	38.52	42.53	**?**	**?**
12:32:00 AM	27.42	32.9	36.33	**?**	**?**
12:37:00 AM	32.1	38.52	42.53	**?**	**?**
12:52:00 AM	65.54	78.65	86.84	**?**	**?**
⋮	⋮	⋮	⋮	⋮	⋮

Table 10. A sample historical training dataset

Time_ monitoring	CPU_utilization	Network_overhead	Memory_usage	VM/Host _state	Fault state
10:12:00 AM	57.22	68.66	99.05	Serious	yes
10:12:05 AM	23.34	28.01	35.02	Normal	no
10:17:00 AM	58.96	70.75	20.12	Minor	no
10:19:02 AM	55.08	66.10	72.98	Minor	no
10:20:25 AM	32.16	38.59	42.61	Normal	no
10:21:03 AM	44.86	53.83	59.44	Minor	no
10:22:20 AM	31.84	38.21	42.19	Normal	no
10:27:70 AM	27.28	32.74	36.15	Normal	no
10:28:11 AM	61.16	73.39	81.04	Serious	yes
10:30:32 AM	38.1	45.72	50.48	Minor	no
⋮	⋮	⋮	⋮	⋮	⋮

Appendix B. A sample java and Scala programs coding

```
public class Training_Filtering_Dataset {
    public static class JobMapper extends Mapper<Object, Text, Text, IntWritable>{
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();
        public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
         String component_name = "";
         String[] row =    value.toString().split(",");
         if(row.length >= 5){
                String state_component=row[3];
                for (int col=0; col<=3; col++) //columns 0-2 contain data about what the components measures
                   {
                      if (col==3 ){
                      state_component=row[3];
                      component_name="State_component";
                      }else if (Double.parseDouble(row[col])>=0 && Double.parseDouble(row[col])<=25){
                      state_component="Normal";
```

```
                }else if (Double.parseDouble(row[col])>=26 && Double.parseDouble(row[col])<=74){
                    state_component="Minor";
                }else if (Double.parseDouble(row[col])<=100 && Double.parseDouble(row[col])>=75){
                    state_component="Serious";}
                    if (col==0)
                    component_name="CPU";
                else if (col==1)
                    component_name="Memory";
                else if(col==2)
                    component_name="Network";
                    word.set(component_name+"-"+state_component+"-"+row[4]);
                    context.write(word, one);
            }}}
public static class JobReducer
                extends Reducer<Text,IntWritable,Text,IntWritable> {
            private IntWritable result = new IntWritable();
            public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
InterruptedException {
                int sum = 0;
                for (IntWritable val : values) {
                      sum += val.get();}
                result.set(sum);
                context.write(key, result);
            }   }
// Scale program for engine
        package streaming
           import org.apache.spark.streaming.{Seconds, StreamingContext}
           object DataStream {
           def main(args: Array[String]) {
           val ssc = new StreamingContext("local[4]", "DataStream", Seconds(3))
           val test_dataset = ssc.textFileStream("hdfs://localhost:9000/result")
           val instances = test_dataset.flatMap(_.split(" "))
           val pairs = instances.map(word => (word, 1))
           val instancesCounts = pairs.reduceByKey(_ + _)
           instancesCounts.print()
          ssc.start()
          ssc.awaitTermination()
    }}
```

Figure 16. A programs Java and Scale coding for training, filtering and streaming dataset

```
public class Classify {
public static void main(String[] args) throws Exception {
      BufferedReader breader=null;
      breader= new BufferedReader(new FileReader("("hdfs://127.0.0.1:9000/Input_dataset-_train"));
      Instances train=new Instances(breader);
      train.setClassIndex(train.numAttributes()-1);
      breader=new BufferedReader(new FileReader("hdfs://127.0.0.1:9000/Input_dataset-_test"));
      Instances test=new Instances (breader);
      test.setClassIndex(train.numAttributes()-1);
      breader.close();
      J48 tree=new J48();
      tree.buildClassifier(train); //build classifier
      Instances labeled= new Instances(test);
```

```
            for (int i=0;i<test.numInstances();i++){ // label instances
            double clsLabel=tree.classifyInstance(test.instance(i));
            labeled.instance(i).setClassValue(clsLabel); }
      BufferedWriter writer=new BufferedWriter(new FileWriter("hdfs://127.0.0.1:9000/out_
spark/out_new_dataset_test.xml "));
      writer.write(labeled.toString());
      writer.close();
      StartNBC.main(args);}}
public class StartNBC{
public static void main(String[] args )throws Exception{
BufferedReader breader =null;
breader =new BufferedReader(new FileReader("c:/algorithms/ result_classify3.xml"));
Instances train=new Instances (breader);
train.setClassIndex(train.numAttributes()-1);
breader.close();
NaiveBayes nB=new NaiveBayes();
nB.buildClassifier(train);
Evaluation eval=new Evaluation(train);
eval.crossValidateModel(nB, train, 20, new Random(1), args);
System.out.print(eval.toSummaryString("\nReaults\n======\n", true));
System.out.println("The F.Measure= "+eval.fMeasure(1)+"%"+"      "+ "The Precision="+eval.precision(1)+"
"+"%"+ "The Recall="+eval.recall(1)+"%"); }}
```

Figure 17. A program coding for combining the testing and training datasets classifier and evaluation results

6. References

[1] Witten, I. H., Frank, E., Hall, M. A., & Pal, C. J., “Data Mining: Practical machine learning tools and techniques,” Morgan Kaufmann, 2016.

[2] Alsheikh, Mohammad Abu, et al., "Mobile big data analytics using deep learning and apache spark," IEEE network, vol. 30 no. 3, pp. 22-29, 2016.

[3] Agrawal, Dakshi, and Vasileios Pappas,"Virtual machine monitoring in cloud infrastructures," U.S. Patent, no. 9, 600,308, 21 Mar. 2017.

[4] Peng, X., Tang, Y., Du, W., & Qian, F. ,"Multimode Process Monitoring and Fault Detection: A Sparse Modeling and Dictionary Learning Method," IEEE Transactions on Industrial Electronics, vol. 64, no.6, pp. 4866-4875, 2017.

[5] Zhang, Tianwei, and Ruby B. Lee. ,"Monitoring and Attestation of Virtual Machine Security Health in Cloud Computing," IEEE Micro, vol. 36, no. 5, pp. 28-37, 2016.

[6] Le, M. K., Chang, H. T., Chang, Y. M., Hu, Y. H., & Chen, H. T., “An efficient multilevel healthy cloud system using Spark for smart clothes,” In Computer Symposium (ICS), 2016 International (pp. 182-186). IEEE, December 2016.

[7] Aceto, G., Botta, A., De Donato, W., & Pescapè, A., ”Cloud monitoring: A survey. Computer Networks,” vol. 57 no. 9, pp. 2093-2115, 2013.

[8] Ajay Kumar, Ravi Shankar, Alok Choudhary & Lakshman S. Thakur, “A big data MapReduce framework for fault diagnosis in cloud-based manufacturing,” International Journal of Production Research, vol. 54, no. 23, pp. 7060-7073, 2016.

[9] Melo, C., Matos, R., Dantas, J., & Maciel, P., “Capacity-Oriented Availability Model for

Resources Estimation on Private Cloud Infrastructure." In Dependable Computing (PRDC), 2017 IEEE 22nd Pacific Rim International Symposium on (pp. 255-260), IEEE, January - 2017.

[10] Zaharia, Matei, et al., "Apache Spark: A unified engine for big data processing," Communications of the ACM, vol. 59, no. 11, pp. 56-65, 2016.

[11] Shyam, R., HB, B. G., Kumar, S., Poornachandran, P., & Soman, K. P. , "Apache spark a big data analytics platform for smart grid," Procedia Technology, vol. 21, pp. 171-178, 2015.

[12] Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., ... & Xin, D., "Mllib: Machine learning in apache spark," The Journal of Machine Learning Research, vol. 17, no.1, pp. 1235-1241, 2016.

[13] Ranjan, Rajiv, "Streaming big data processing in datacenter clouds," IEEE Cloud Computing, vol. 1, no. 1, pp. 78-83, 2014.

[14] Kotsiantis, Sotiris B., I. Zaharakis, and P. Pintelas., "Supervised machine learning: A review of classification techniques," pp. 3-24, 2007.

[15] Gupta, "A. Learning Apache Mahout Classification," Packt Publishing Ltd; Feb - 2015.

[16] Box, George EP, Gwilym M. Jenkins, Gregory C. Reinsel, and Greta M. Ljung, "Time series analysis: forecasting and control," John Wiley & Sons, 2015.

[17] Krishnan, Venkatarama, and Kavitha Chandra, "Probability and random processes," John Wiley & Sons, 2015.

[18] Ameen Alkasem, Hongwei Liu, and DechengZuo, "Utility Cloud: A Novel Approach for Diagnosis and Self-healing Based on the Uncertainty in Anomalous Metrics," In Proceedings of the 2017 International Conference on Management Engineering, Software Engineering and Service Sciences (ICMSS '17), Yulin Wang (Ed.), pp. 99-107, ACM, New York, NY, USA, 2017.

[19] Chisnall, David, "The definitive guide to the xen hypervisor," Pearson Education, 2008.

[20] Massie, Matthew L., Brent N. Chun, and David E. Culler, "The ganglia distributed monitoring system: design, implementation, and experience," Parallel Computing, vol. 30, no. 7, pp. 817-840 2004.

[21] Wang, Chengwei, Vanish Talwar, Karsten Schwan, and ParthasarathyRanganathan, "Online detection of utility cloud anomalies using metric distributions," In Network Operations and Management Symposium (NOMS), 2010 IEEE, pp. 96-103. IEEE, 2010.

[22] Fujii, Keisuke, "Extended kalman filter," Refernce Manual, 2013.

[23] Doane, David P., and Lori E. Seward, "Applied statistics in business and economics," USA: Irwin, 2005.

[24] Gawrilow, Ewgenij, Simon Hampe, and Michael Joswig, "The polymake XML file format," In International Congress on Mathematical Software, pp. 403-410. Springer International Publishing, 2016.

[25] Alkasem A, Liu H, Decheng Z, Zhao Y. "AFDI: A Virtualization-based Accelerated Fault Diagnosis Innovation for High Availability Computing. arXiv preprint arXiv:1507.08036., Jul 2015.

[26] A Ameen, H Liu, M Shafiq , and D Zuo, "A New Theoretical Approach: A Model Construct for Fault Troubleshooting in Cloud Computing," Mobile Information Systems, Article ID

9038634, 6 September 2017.

[27] Mertler, Craig A., and Rachel Vannatta Reinhart, “Advanced and multivariate statistical methods: Practical application and interpretation,” Routledge, 2016.

[28] Alkasem, Ameen, and Hongwei Liu, "A Survey of Fault-tolerance in Cloud Computing: Concepts and Practice," Research Journal of Applied Sciences, Engineering and Technology, vol. 11, no. 12, pp. 1365-1377, 2015.

Acknowledgment

We are also thankful to anonymous reviewers for their valuable feedback and comments for improving the quality of the manuscript.