An Ensemble MIC-based Approach for Performance Diagnosis in Big Data Platform

Pengfei Chen, Yong Qi, Xinyi Li, Li Su
School of Electronic and Information Engineer, Xi'an Jiaotong University
Email: fly.bird.sky@stu.xjtu.edu.cn, qiy@mail.xjtu.edu.cn, ,xingga.li,suli@stu.xjtu.edu.cn

***Abstract*—The era of big data has began. Although applications based on big data bring considerable benefit to IT industries, governments and social organizations, they bring more challenges to the management of big data platforms which are the fundamental infrastructures due to the complexity, variety, velocity and volume of big data. To offer a healthy platform for big data applications, we propose a novel signature-based performance diagnosis approach employing MIC invariants between performance metrics. We formalize the performance diagnosis as a pattern recognition problem. The normal state of a big data application is used to train a set of MIC (Maximum Information Criterion) invariants. One performance problem occurred in the big data application is identified by a unique binary tuple consisted by a set violations of MIC invariants. All the signatures of performance problems form a diagnosis knowledge database. If the KPI (Key Performance Indicator) of the big data application deviates its normal region, our approach can identify the real culprits through looking for similar signatures in the signature database. To detect the deviation of the KPI, we propose a new metric named *unpredictability* based on ARIMA model. And considering the variety of big data applications, we build an ensemble performance diagnosis approach which means a unique ARIMA model and a unique set of MIC invariants are built for a specific kind of application. Through experiment evaluation in a controlled environment running a state of the art big data benchmark, we find our approach can pinpoint the real culprits of performance problems in an average 83% precision and 87% recall which is better than a correlation based and single model based performance diagnosis.**

***Keywords*-Big data; Hadoop; performance diagnosis; MIC; ARIMA**

I. Introduction

As the exponential growth of data in our daily life, we have stepped into the big data era. According to [1], 2.5 quintillion bytes of data are created everyday and the increasing trend will be remained in the foreseeable future. Computer scientists, physicists, economists, mathematicians, political scientists, bio-informaticists, sociologists, and even musicians are eager to embrace the massive quantities of information produced by and about people, things, and their interactions. To mine the valuable information in the scrambled data piles, people from academia and industry have conducted numerous work. In the academic world, researchers have designed and developed several big data benchmarks to understand the characteristics of big data applications [2], [3], [4]. In the industrial world, big data based applications such as business intelligence and predictive analyses are implemented to analyze the customer behaviors or predict the data trend. It's reasonable to believe big data becomes a new driving force to change the world.

However due to the instinct complexity and three 'v' (i.e. velocity, volume and variety) properties of big data [2], [5], several new challenges are brought to the management of big data platforms. First, different from the transaction-based applications, the execution duration of one request issued to the big data platform is long always ranging from several minutes to a few hours and even longer. Therefore the QoS metrics which are common used in transaction-based applications like response time of a request or throughput may not work any more. A new metric to reveal the health state of the big data application is urgently needed. Second, the performance models under different types of big data applications are not identical. Hence, it's necessary to build a unique performance model for a unique type of workload. Considering the aforementioned two points, we use the metric *unpredictability* as the health indicator of the big data application and build an ensemble performance diagnosis approach to adapt to the workload changes.

The large cardinality of suspicious cause set hinders us to uncover the actual culprits precisely and completely. Therefore it's impractical to propose a silver bullet to resolve all the performance problems. In this paper we restrict our diagnosis on a subset of performance problems in the big data platform based on Hadoop [6]. From previous studies [7], [8], [9], we find out performance problems are partially caused by the runtime environment changes (e.g. resource hogs and configuration changes). And after reviewing the bugs of Hadoop software [10], we observe that large number of bugs can cause performance problems. Here we only take into account the bugs relevant to the abnormal consumption of physical resources (e.g. CPU) or logical resources (e.g. lock). The reasons of choosing these bugs are: these metrics can be readily collected at runtime without instrumenting source code; large number of these bugs exist in the software.

Numerous previous work has been done in this area, but they mainly put their emphasis on locating anomaly coarsely (e.g. at service level [11], [12] or VM level[8], [9]) instead of identifying the real reasons causing performance problems. [15] is the most close to our work, but it leverages Pearson correlation coefficient to construct the correlation relationship between two performance metrics which leads to some inefficiency to performance diagnosis. Our aim is to identify the real causes of performance problems at fine granularity such as CPU hog, configuration errors or software bugs in a local host.

To fulfill this task, we formalize the performance diagnosis as a pattern recognition problem and propose a novel ensemble MIC-based performance diagnosis approach. MIC (i.e. Maximum Information Criterion) [13] is adopted to discover the associative relationship between two variables. The basic idea is to build a signature database where every item is mapped to one kind of performance problem and when performance anomaly occurs, the culprits are pinpointed by looking for a similar signature in the signature database. We first use the normal running of each type of big data workload to train a performance model (i.e. the parameters of ARIMA [14] model) and a set of MIC invariants. Then we build a signature for each performance problem. The signature is expressed as a binary tuple consisted by the violations of MIC invariants. All the signatures form a knowledge database which will be used to find the culprits in the future performance diagnosis. If a performance anomaly is detected using ARIMA model, the performance diagnosis is triggered. Then we calculate the MIC scores under the abnormal environment and find out all the violations among all the MIC scores. Finally we find the real culprits of performance problems by looking for a similar signature in the signature database. Through the experiment evaluation in a state of the art big data benchmark, we find out our approach can detect the real culprits of performance problems in an average 83% precision and 87% recall which is better than a correlation based and single-model based performance diagnosis.

Our contributions are three-fold:

- We propose a new performance anomaly detection method (i.e. unpredictability) based on ARIMA model for big data applications.
- We introduce a signature-based approach employing MIC invariants to correlate a specific kind of performance problem with a binary tuple which is consisted by a set of violations of MIC invariants.
- Considering the variety of the big data application, we propose an ensemble approach (i.e. building the ARIMA model and the set of MIC invariants for different kind of big data workload respectively) to diagnose the real causes of performance problems in big data platform. The experimental results show that this approach can find out the culprits accurately.

The rest of this paper is organized as follows. Section II depicts the basic idea and problem formulation of our approach. Section III demonstrates the details of our approach. Section IV shows the experimental evaluation in a state of the art benchmark. Section V concludes this paper.

II. Problem Formulation

The intuition underling our approach is that the performance problems occurred in the big data platform can manifest themselves as the violations upon the correlation coefficients between performance metrics which is also mentioned in [15]. In the normal situations, the correlation coefficients stay in a particular region. But in the abnormal situations, the value of correlations may be deviated. Grasping this point, we build a unique signature for a unique performance problem using the violations of the correlations and formalize the performance diagnosis as a pattern recognition problem. Assuming $(M_1, M_2, \cdots, M_m)$ represents a set of performance metrics, the matrix COR where each entry Cor_{M_i,M_j} denotes the correlation coefficients between metric M_i and metric M_j, $i \neq j$ represents all the correlation coefficients between performance metrics, we construct the correlation matrixes COR_{normal} and $COR_{abnormal}$ for the normal and abnormal situation respectively. If $|Cor'_{M_i,M_j} - Cor_{M_i,M_j}| \geq \varepsilon$, a violation occurs, where Cor'_{M_i,M_j} denotes the entry in matrix $COR_{abnormal}$, Cor_{M_i,M_j} denotes the entry in matrix COR_{normal} and is called a *correlation invariant*, ε is the preset threshold, say $\varepsilon = 0.2$ in this paper. We calculate all the violations and use a binary tuple $(1, 0, 1, 0, 1, \cdots, 0)$ where '0' implies no violation and '1' implies violation to identify a performance problem uniquely. The length of the tuple is the number of entries in the correlation matrix COR. Integrating all the binary tuples constructed in multiple abnormal situations, we obtain a signature database which will be used in the future performance diagnosis. And considering the variety of big data workload, we will build a correlation matrix COR_{normal} for each kind of workload. Although Pearson correlation coefficient [16] is widely used to determine the correlation between two variables, this paper introduces a state of the art and more efficient correlation detection method named MIC [13]. If a performance anomaly is detected in a Hadoop node (a name node or data node) , the performance diagnosis is triggered. We first calculate the violation tuple under the current abnormal situation then find a similar signature in the signature database. If a similar signature is found, the culprit is pinpointed otherwise we leave the problem to the system administrators who will manually check the problems. Once the performance problems is resolved, a new signature will be added into the signature base.

Our approach mainly includes two parts and four modules shown in Figure 1. The offline part contains two modules: performance model building and signature base building. The former module is to determine the parameters of a predictive model based on ARIMA for specific types of workload. While the latter module is to train the signatures for different performance problems using MIC under different workload. The online part also contains two modules: performance anomaly detection and cause inference. The anomaly detection module utilizes the performance model generated by the performance model building module to detect the performance violations. If an anomaly is detected, the cause inference is triggered to analyze the real culprits leveraging the signature base generated by the signature base building module. We will give the details of all the modules in the following section.

III. Details of Performance Diagnosis

A. Performance Model Building

Intuitively the performance metrics (e.g. CPU, memory) of one application under normal running could be described by a dynamic process model. In this paper we adopt a

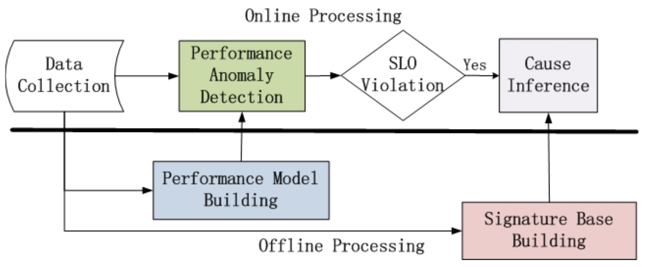

Fig. 1. The framework of our performance diagnosis approach

commonly used predictive model, ARIMA [14], to depict the dynamics of the performance metrics. ARIMA models are, in theory, the most general class of models for forecasting a time series which can be stationarized by transformations such as differencing and lagging. The acronym ARIMA stands for "Auto-Regressive Integrated Moving Average", always denoted by $ARIMA(p, d, q)$, where p is the order of "auto-regressive" terms, q is the order of "moving average" terms and d is the difference order. The ARIMA model is built on the ARMA [14]. In the original time series, a seasonal or non-seasonal trend may exist which makes the time series non-stationary. However a fundamental assumption for ARMA is that the time series is stationary. Therefore to employ ARMA model, the original time series is differenced to make it stationary. Let $X(t)$ represent a time series of one performance metric. We first use *runs test* [17] to validate whether the time series is stationary, if not stationary, the difference process is conducted. The difference procedure is formalized as : $X'(t) = (1 - B)^d X(t)$ where B is the back shift operator: $BX(t) = X(t - 1)$, d is the difference order, when $d = 1$, $X'(t) = X(t) - X(t - 1)$. After that, we use *zero-average* to reprocess the differenced time series that is: $Y(t) = X(t) - \bar{X}(t)$ where $Y(t)$ is the preprocessed time series and $\bar{X}(t)$ is the average of the time series X_t.

The following step is to build the $ARMA(p, q)$ model for $Y(t)$: $\phi(B)Y(t) = \theta(B)a(t)$ where $\phi(B)$ is the autoregressive operator, represented as a polynomial in the back shift operator defined as: $\phi(B) = 1 - \phi_1 B - \cdots - \phi_q B^p$, $\theta(B)$ is the moving-average operator, represented as a polynomial in the back shift operator defined as:$\theta(B) = 1 - \theta_1 B - \cdots - \theta_q B^q$ and a_t is the independent disturbance, also called the random error which is always define as a white noise with zero average and variance σ^2.

The critical steps to determine the ARMA model are model identification (i.e. determine the parameters p and q) and parameter estimation (i.e. determine the parameters $(\theta_1, \theta_2, \cdots, \theta_p)$ and $(\phi_1, \phi_2, \cdots, \phi_q)$). We adopt the the method mentioned in [18] to determine p and q. The auto correlation function (ACF) and partial auto correlation function (PACF) are calculated for $ARMA(p, q)$. If the ACF and PACF of $ARMA(p, q)$ exhibit a sharp decrease (i.e. trails off) or vibrate around y-axis, then p and q are determined. After determine the model identification, we use the Least-Square method to estimate the parameters $(\theta_1, \theta_2, \cdots, \theta_p)$ and $(\phi_1, \phi_2, \cdots, \phi_q)$. Finally we obtain the performance model for one performance metric under one kind of specific big data workload. According to our observation, there is no single $ARIMA(p, d, q)$ model to model the performance of all kinds of big data workload. Hence, we develop an ensemble model to build a unique $ARIMA(p, d, q)$ for each specific workload.

B. Performance Anomaly Detection

After the performance models are established, they will be leveraged to conduct performance anomaly detection. Our basic idea for anomaly detection is that in abnormal situations the performance models trained in the normal situations are violated, we define this violation as *unpredictability*. Formally, $(M_1, M_2, \cdots, M_m)$ represents the performance metrics collected from the big data platform at runtime, $M'_i(t)$ is predicted by the $ARIMA$, $M_i(t)$ is the real observation of metric M_i at time t, the absolute value of deviation between $M'_i(t)$ and $M_i(t)$ is defined as the *unpredictability* denoted by $\xi(t)$: $\xi(t) = |M'_i(t) - M_i(t)|$. If $\xi(t) > \alpha$, α is a preset threshold, a performance anomaly occurs. Here we adopt the simple threshold based anomaly detection method, but more comprehensive methods such as likelihood ratio test [18] are encouraged to be applied. Considering the computation complexity, we only use four performance metrics (i.e. $m = 4$): CPU utilization, free memory, disk queue length and network bandwidth as the performance indicators to conduct performance anomaly detection.

C. Signature Database Building

As mentioned in Section II, we build a unique signature for each performance problem occurred in the big data platform. The signature is expressed as a binary tuple which is consisted by a set of violations of correlation coefficients between performance metrics. To capture the correlation between a pair of performance metrics, we introduce a state of the art method named MIC. The conventional *Pearson correlation coefficient* and R^2 based correlation detection method can precisely capture the linear relationships or the functional relationships, but are weak in capturing the non-linear or non-functional relationships between two variables. While, MIC shows great power to capture the complex correlation relationships. To keep the paper self-contained, we give several definitions and preliminaries about MIC.

Definition 1. *Resolution Grid: Given a finite set D of ordered pairs, the x-values of D is partitioned into x bins and the y-values of D is partitioned into y bins, allowing empty bins. Such a pair of partitions an $x - by - y$ resolution grid.*

Given a grid G, let $D|_G$ be the distribution induced by the points in D on the cells of G. For a fixed D, different grids G result in different distributions $D|_G$.

Definition 2. *Mutual Information: The mutual information between two random variables X and Y is defined as:*

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) log(\frac{p(x,y)}{p(x)p(y)})$$

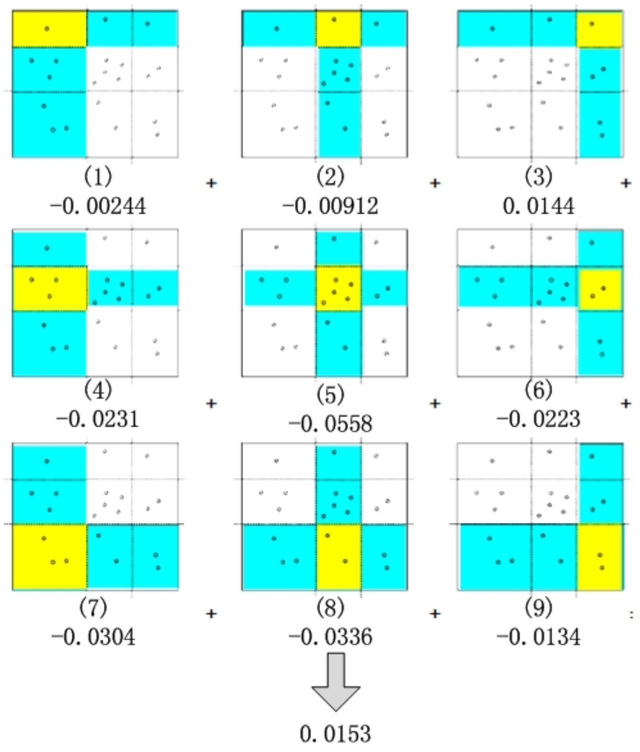

Fig. 2. The calculation of Mutual Information when $x = 3, y = 3$

where $p(x, y)$ is the joint probability distribution function, $p(x)$ and $p(y)$ are the marginal probability distribution function.

In the grid D, $p(x)$, $p(y)$ and $p(x, y)$ are the ratio of the number of data points falling in the relevant cell and the total data points. For instance, in Figure 2 when $x = 2, y = 2$, $p(x) = 8/20 = 0.4$, $p(y) = 10/20 = 0.5$, $p(x, y) = 5/20 = 0.25$ and when D is partitioned into $3-by-3$ grid, $I(X, Y) = (-0.00244)+(-0.00912)+0.0144+(-0.0231)+(-0.0558)+(-0.0223) + (-0.0304) + (-0.0336) + (-0.0134) = 0.0153$

Definition 3. *Highest Mutual Information: For a finite set $D \subset R^2$ and positive integers x, y, define*

$$I^*(D, x, y) = maxI(D|_G)$$

where the maximum is over all grids G with x columns and y rows, and $I(D|_G)$ denotes the mutual information of $D|_G$.

Definition 4. *Characteristic Matrix: The characteristic matrix $M(D)$ of a set D of two-variable data is an infinite matrix with entries:*

$$M(D)_{x,y} = \frac{I^*(D, x, y)}{log \quad min\{x, y\}}$$

Definition 5. *Maximal information coefficient (MIC):The Maximal Information Coefficient (MIC) of a set D of two-variable data with sample size n and grid size less than $B(n)$ is given by:*

$$MIC(D) = max_{xy<B(n)}\{M(D)_{x,y}\}$$

where $\omega(1) < B(n) \leq O(n^{1-\varepsilon})$ for some $0 < \varepsilon < 1$

Just as pointed in [13], we also use $B(n) = n^{0.6}$. The detailed description of the definitions and the boundary of some variables could be found in the supporting online material of [13].

The procedure to calculate MIC includes the following steps:

- step 1: Find the approximative highest mutual information for data D, namely $I^*(D, x, y)$.The core of this step is to find a optimal $x - axis$ partition given fixed $y - axis$ partition using dynamic programming (Refer to the supporting online material of [13] for details).
- step 2: Construct the characteristic matrix using the obtained $I^*(D, x, y)$ according to the definition.
- step 3: Calculate the MIC for D by looking for the maximum value in the characteristic matrix.

The whole procedure is shown in algorithm 1.

Algorithm 1 MIC(D)

```
Require: $D$ is a set of ordered pairs
Require: $B$ is an integer greater than 3
  for all $(x, y)$ such that $xy \leq B$ do
    $I_{x,y} \leftarrow ApproxMAXMI(D, x, y)$
    // find the approximative highest mutual information
    $M_{x,y} \leftarrow \frac{I_{x,y}}{min\{logx,logy\}}$
    // construct the characteristic matrix
  end for
  $MIC(D) \leftarrow max\{M_{x,y}\}$
```

In the normal situations, we build a correlation matrix COR_{normal} for a set of performance metrics using MIC. Each entry in COR_{normal} is a correlation score between a pair of metrics named *MIC invariant*. For each performance problem (e.g. memory leak, cpu hog, network hog) occurred in the big data platform, we reproduce them and build the correlation matrix $COR_{abnormal}$ under the abnormal situations. Then we compare the matrixes COR_{normal} with $COR_{abnormal}$ and find out all the violations according to the description in Setion II. Finally a binary tuple, a signature, is obtained for each performance problem. Integrating all the signatures, a signature database is constructed. Considering the differences of MIC invariants under different kinds of workload, we will train a COR_{normal} for each kind of workload.

D. Cause Inference

After the performance model and the signature database are constructed, the performance diagnosis can be ready to work. If a performance anomaly using the approach mentioned in the former section, the performance diagnosis is triggered. Then we calculate the correlation matrix $COR_{abnormal}$ and find out all the violations by comparing $COR_{abnormal}$ with COR_{normal} under the same kind of workload. Assume $S_a = (1, 0, 1, \cdots, 1)$ represents the binary tuple of violations when a performance anomaly occurs, $S_d = (0, 1, 1, \cdots, 0)$ represents a signature in the signature database, we use the

Hamming distance to denote the similarity between S_a and S_d, $D_h = Hamming(S_a, S_b)$. If $D_h < \varepsilon$, the cause with respect to the signature is chosen as the final culprit of the current performance problem where ε is the preset threshold, say $\varepsilon = 3$ which functions well in this paper; otherwise the performance problem is left to the system administrators who will manually check the problems. Once the performance problems is resolved, a new signature will be added into the signature base.

IV. Experiment Evaluation

We have implemented a prototype and deployed it in the controlled environment. We use an off-the-shelf tool, *collectl* [19], to collect the process and operating system performance metrics. The collected metrics not only include coarse-grained CPU, memory,disk and network utilization but also the fine-grained metrics such as CPU context switch per second, memory page faults, etc. The number of performance metrics is 30 and the sample interval is 5 second. In the following, we will give the details of our experimental methodology and evaluation results in two benchmarks: BigDataBench [20].

A. Evaluation Methodology

Due to the lack of real operating platforms, our approach is only evaluated in a controlled distributed system. But we believe it works well in a real system without exceptions. The controlled system contains five physical server machines hosting the benchmark. Each physical machine has a 8-core Xeon 2.1 GHZ CPU and 16GB memory and is virtualized into five VM instances including domain 0 by KVM. Each VM has two vcpu and 2GB memory and runs a 64-bit CentOS 6.2, Hadoop 1.0.2, Mahout 0.6 and our prototype.

BigDataBench is a benchmark suite from web search engines, to benchmark and rank systems that are running big data applications. It provides 6 representative applications from search engines which are the most important domain in Internet services in terms of the number of page views and daily visitors. It also provides an innovative data generation tool to generate scalable volumes of big data from a small-scale of real data preserving semantics and locality of the real data. In this paper we only choose four kinds of applications based on Hadoop in BigDataBench: Sort, Wordcount, Grep and Naive Bayesian classifier and leave other applications for the future work. During all the experiments, the volume of the big data is 10G generated by the tool in BigDataBench benchmark. For the performance problems caused by runtime environment changes, we inject the following faults: 1) CpuHog: a CPU-bound application co-locates with TaskTracker competing for CPU resource; 2) MemLeak: a memory-bound application continually consumes memory of the data node; 3) DiskHog: we use a disk-bound program to generate a mass of disk reads and writes on the data node; 4) NetworkJam: we use "tc", a traffic control tool in Linux, to mimic the packet loss on name node; for the performance problems caused by software bug, we inject the following faults: 1) CpuBug: we write a new java class which largely consumes CPU resource and load it in the JVM in the data node using the attach API in J2EE 6.0; 2) MemBug: a new class is injected in the jvm using the same method as the CpuBug. 4KB memory will be leaked once this class is executed.

Each of the faults mentioned above will be repeated for 20 times and last 5 minutes. And multiple faults may be simultaneously injected in one node. To get the ground truth, we will log the fault injection time and types. We leverage two commonly used metrics: precision and recall to evaluate the effectiveness of our system.

$$Recall = \frac{N_{tp}}{N_{tp} + N_{fn}}, Precision = \frac{N_{tp}}{N_{tp} + N_{fp}}$$

where N_{tp}, N_{fn}, N_{fp}, and N_{tn} denote the number of true positives, false negatives, false positives, and true negatives, respectively.

B. Performance Model Building

In this section we conduct several experiments to answer the questions "how to build the ARIMA model", "why an ensemble model is necessary" and "is it efficient to detect performance anomaly using *unpredictability*".

Due to the limited space, we only show the experiment results on CPU total utilization metric under different kinds of applications. Figure 3 shows the CPU utilization of Sort, Wordcount, Gred and Naive Bayesian classifier application. From this figure, we observe that the characteristics of CPU utilization for these four applications are apparently different. Especially a two-phase feature is manifested in Sort application. Take the CPU utilization of Sort application for example, we calculate the ACF and PACF of the original data series and observe that the ACF shows a slow decreasing with respective to the lag (shown in Figure 4 (a) which implies that the parameters p, d and q need further adjustion. When $p = 3$, $d = 1$, $q = 2$, we observe a significant trail off in ACF and vibration in PACF therefore there is no need to adjust p (see Figure 5), d and q any more. Finally we get the ARIMA model for the CPU utilization of Sort application: $\phi(B) = 1 + 0.7772B + 0.6869B^2 - 0.1509B^3, \theta(B) = 1 - 1.2717B - B^2$, the variance of $a(t)$ is 24.65.

We use the same method to model the CPU utilization of wordcount application and obtain the model $ARIMA(1,1,1)$: $\phi(B) = 1 - 0.3122B, \theta(B) = 1 + B$, the variance of $a(t)$ is 0.5596. Comparing this model with the model obtained in Sort application, we can see they are significantly different. Therefore it's necessary to build a unique ARIMA model for a specific application.

To validate the effectiveness of performance anomaly detection using *unpredictability*, we use the $ARIMA(3,1,2)$ model trained for CPU utilization of Sort application under normal situation to calculate the *unpredictability* of the CPU utilization under CpuHog fault injection and observe the following result shown in Figure 6. A CpuHog fault is injected at time point 37 on x-axis, before this time point we observe that the predicted CPU utilization is consistent with the original CPU utilization and the *unpredictability*

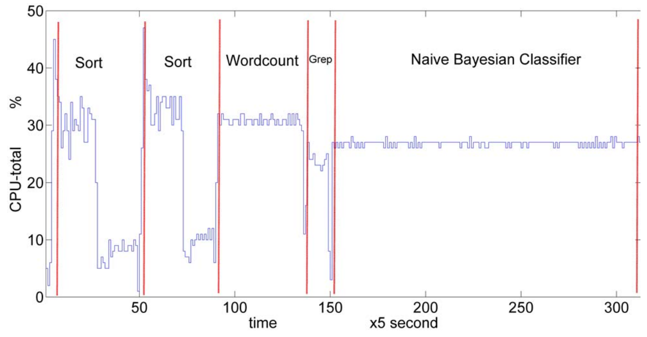

Fig. 3. The CPU utilization under different kinds of workload

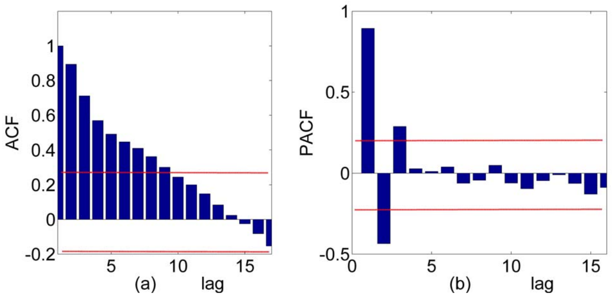

Fig. 4. The ACF and PACF of original CPU utilization

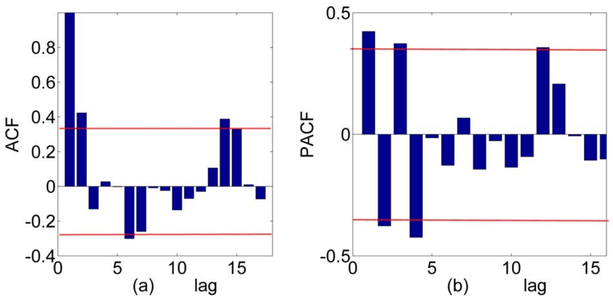

Fig. 5. The ACF and PACF of $ARIMA(3, 1, 2)$

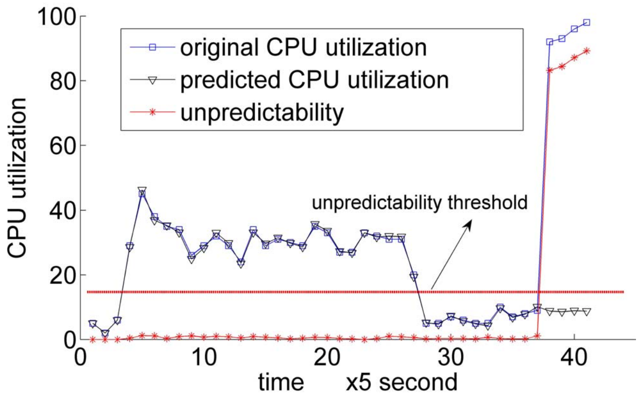

Fig. 6. The performance anomaly detection on CPU utilization

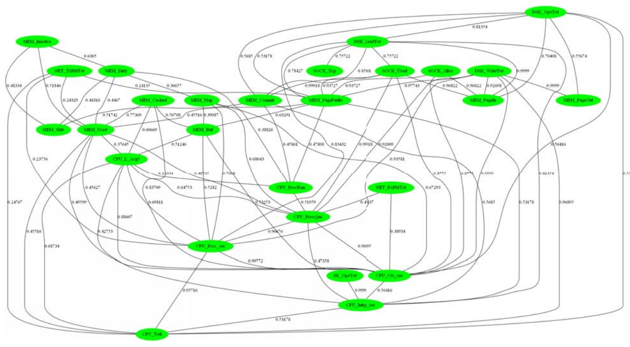

Fig. 7. The correlation pairs under normal running

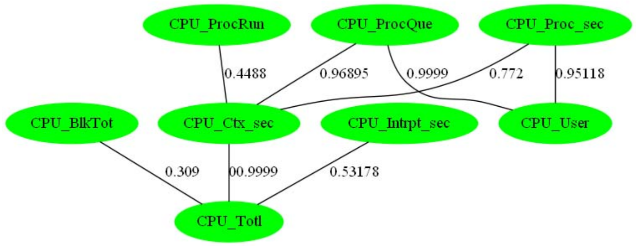

Fig. 8. Zoom out of Figure 7

stays under the unpredictability threshold; after this time point the predicted result deviates from the original data sharply and the *unpredictability* exceeds the unpredictability threshold which indicates a performance anomaly occurs. Apparently the unpredictability threshold can affect the performance anomaly detection. In this paper we preset the threshold statically, say 17 for CPU utilization, which may bias the precision and recall of the performance diagnosis a little and the dynamical regulation of this threshold will be discussed in our future work.

C. Signature Building

30 performance metrics are collected in our prototype and the MIC scores of 435 $(30 * (30 - 1)/2)$ pairs of performance metrics are calculated. The correlation relationship with a very small MIC score carries no information to build the signature for a performance problem, hence we eliminate this correlation. Take the Sort application for example, under normal running we obtain a set of 83 correlation pairs shown in Figure 7 and Figure 8 when the threshold of MIC score is set 0.1. These correlation pairs are used as MIC invariants to build the signatures. Figure 9 shows the correlation pairs in the CpuHog fault injection experiment. The red line in Figure 9 denotes the violated correlation pairs, in this experiment we observe 13 violations and nearly all the violations are relevant to CPU metrics. Therefore CpuHog fault can be represented by a tuple $(1, 0, 1, \cdots, 0, 1)$ where '1' denotes the violated MIC invariant.

D. Cause Inference

Figure 10 demonstrates the diagnosis result when only one type of faults is injected in the benchmark. We observe that the precision and recall of most types of faults fall in the range $(80, 90)$ except the MemBug fault. The precision and recall of MemBug fault are only 71% and 67 % respectively. After investigating the $COR_{abnormal}$ under MemBug fault

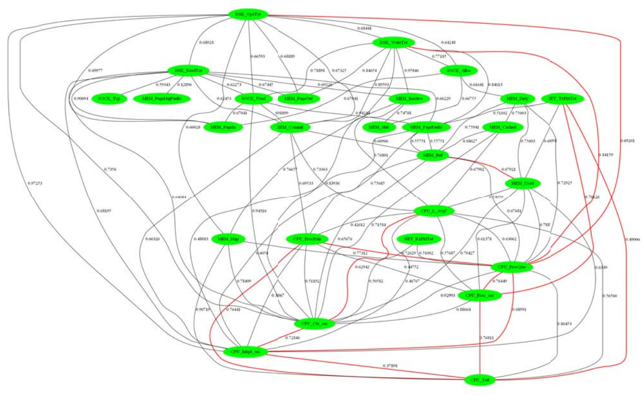

Fig. 9. The correlation pairs under CpuHog fault injection

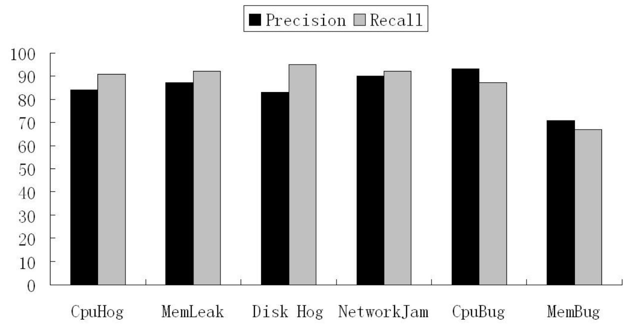

Fig. 10. The diagnosis result when one type of fault is injected

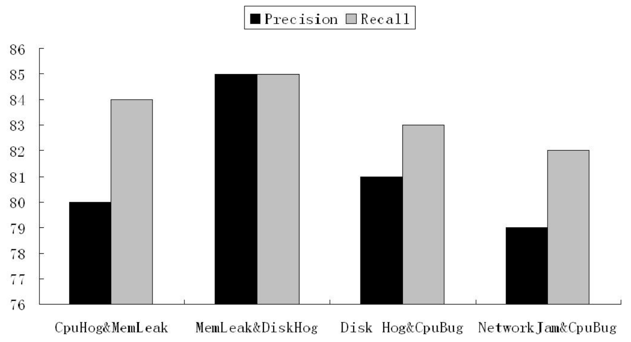

Fig. 11. The diagnosis result when multiple types of faults are injected

injection, we found that the $COR_{abnormal}$ are quite different which means the signature we used can not uniquely represent this kind of performance problem. The improvement of our approach towards MemBug fault will be discussed in our future work. Figure 11 shows the diagnosis result when multiple types of faults are injected in the benchmark simultaneously. Pay attention to that, when multiple faults are injected the signature is built for the combination of these faults instead of single fault. Compared with the result of one type of faults, the average precision and recall (i.e. Average the precision and recall of all types of faults) have a slight decrease about 5% and 9% respectively.

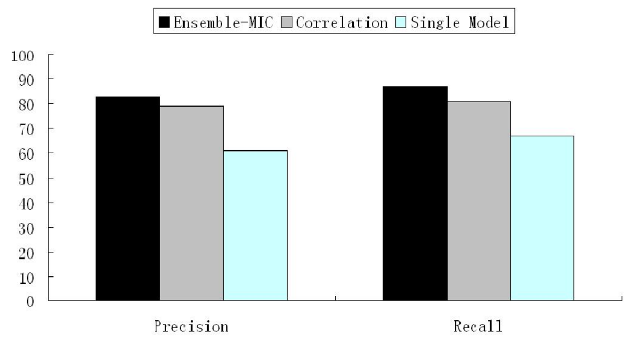

Fig. 12. The comparison when one type of faults is injected

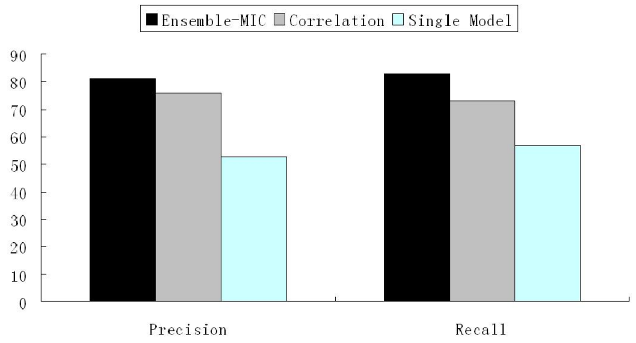

Fig. 13. The comparison when multiple types of faults are injected

E. Comparison

To fully evaluate our approach, we conduct several experiments to compare it with the Pearson correlation coefficient based method which adopt Pearson correlation coefficient to determine the correlation between two performance metrics and the single model based method which doesn't consider the type of applications and trains only one ARIMA model and one set of MIC invariants. The results are shown in Figure 12 and Figure 13. From these two figures, we observe that our approach are better than the other two methods no matter in average precision and average recall. The Pearson correlation coefficient based method may loose several correlations due to the weak power to determine a non-linear or non-functional relationships. In our experiment, we find the relationships between CPU utilization and other Memory metrics are lost leading to a precision decrease in the final performance diagnosis result. The single model based method exhibits the worst performance. There are mainly two reasons: a). a weak power in performance anomaly detection due to the inefficiency of a unique ARIMA model; b). a low quality of the signature database where each item can not represent the performance problem uniquely.

V. Conclusion and Future Work

This paper proposes an ensemble MIC-based approach to pinpoint the culprits of performance problems in the big data platform. The basic idea of the approach is to formalize the diagnosis performance as a pattern recognition problem. We build a unique signature for every performance problem such as CpuHog and use this signature to find the cause of the performance problem in the future performance diagnosis. The signature is expressed as a binary tuple which is consisted by the violations of MIC invariants. If the KPI of the big data application deviates its normal region, our approach can identify the real culprits through looking for similar signatures in the signature database. To detect the deviation of the KPI, we propose an new metric named *unpredictability* based on ARIMA model. And considering the variety of big data applications, we build an ensemble performance diagnosis approach which means a unique ARIMA model and a set of MIC invariants are built for a specific kind of workload. Through experiment evaluation in a controlled environment running a state of the art big data benchmark, we find our approach can pinpoint the real culprits of performance problems in an average 83% precision and 87% recall which is better than a correlation based and single model based performance diagnosis. In the future work, we will discuss a dynamical threshold regulation algorithm for the threshold mentioned in paper such as the *unpredictability* threshold to provide better diagnosis results.

ACKNOWLEDGMENT

This work gets great help from Professor Jianfeng Zhan, a faculty of Institute of Computing Technology of Chinese Academy of Sciences. We show great thanks to him. And this work is supported by the project Grant No. 60933003 sponsored by National Science Foundation of China. The authors would like to thank the anonymous reviewers for their insightful comments.

REFERENCES

[1] [Online]. Available: http://www-01.ibm.com/software/data/bigdata/

[2] W. Gao, Y. Zhu, Z. Jia, C. Luo, L. Wang, Z. Li, J. Zhan, Y. Qi, Y. He, S. Gong *et al.*, "Bigdatabench: a big data benchmark suite from web search engines," *arXiv preprint arXiv:1307.0320*, 2013.

[3] Y. Chen, "We don't know enough to make a big data benchmark suite-an academia-industry view," *Proc. of WBDB*, 2012.

[4] H. Xi, J. Zhan, Z. Jia, X. Hong, L. Wang, L. Zhang, N. Sun, and G. Lu, "Characterization of real workloads of web search engines," in *Workload Characterization (IISWC), 2011 IEEE International Symposium on*. IEEE, 2011, pp. 15–25.

[5] [Online]. Available: http://www.sas.com/big-data/

[6] [Online]. Available: http://www.hadoop.apache.org/

[7] S. Kandula, R. Mahajan, P. Verkaik, S. Agarwal, J. Padhye, and P. Bahl, "Detailed diagnosis in enterprise networks," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, pp. 243–254, 2009.

[8] H. Nguyen, Z. Shen, Y. Tan, and X. Gu, "Fchain: Toward black-box online fault localization for cloud systems," in *Distributed Computing Systems (ICDCS), 2013 IEEE 33nd International Conference on*. IEEE, 2013.

[9] H. Nguyen, Y. Tan, and X. Gu, "Pal: P ropagation-aware a nomaly l ocalization for cloud hosted distributed applications," in *Managing Large-scale Systems via the Analysis of System Logs and the Application of Machine Learning Techniques*. ACM, 2011, p. 1.

[10] [Online]. Available: http://www.issues.apache.org/jira/browse/HADOOP

[11] X. Chen, M. Zhang, Z. M. Mao, and P. Bahl, "Automating network application dependency discovery: Experiences, limitations, and new solutions." in *OSDI*, vol. 8, 2008, pp. 117–130.

[12] P. Barham, R. Black, M. Goldszmidt, R. Isaacs, J. MacCormick, R. Mortier, and A. Simma, "Constellation: automated discovery of service and host dependencies in networked systems," *TechReport, MSR-TR-2008-67*, 2008.

[13] D. N. Reshef, Y. A. Reshef, H. K. Finucane, S. R. Grossman, G. McVean, P. J. Turnbaugh, E. S. Lander, M. Mitzenmacher, and P. C. Sabeti, "Detecting novel associations in large data sets," *science*, vol. 334, no. 6062, pp. 1518–1524, 2011.

[14] P. J. Brockwell and R. A. Davis, *Introduction to time series and forecasting*. Taylor & Francis US, 2002.

[15] B. Sharma, P. Jayachandran, A. Verma, and C. R. Das, "Cloudpd: Problem determination and diagnosis in shared dynamic clouds," in *IEEE DSN*, 2013.

[16] [Online]. Available: http://www.en.wikipedia.org/wiki/Correlation_and_dependence

[17] K. Yuerekli, A. Kurunc, and F. Oeztuerk, "Testing the residuals of an arima model on the cekerek stream watershed in turkey," *Turkish Journal of Engineering & Environmental Sciences*, vol. 29, no. 2, pp. 61–74, 2005.

[18] W. Linping, M. Dan, G. Wen, and Z. Jianfeng, "A proactive fault-detection mechanism in large-scale cluster systems," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*. IEEE, 2006, pp. 10–pp.

[19] [Online]. Available: http://www.collectl.sourceforge.net/

[20] [Online]. Available: http://www.prof.ict.ac.cn/BigDataBench/