InvarNet-X: A Comprehensive Invariant Based Approach for Performance Diagnosis in Big Data Platform

Pengfei Chen(✉), Yong Qi, Di Hou, and Huachong Sun

School of Electronic and Information Engineering, Xi'an Jiaotong University,
Xianning West Road No.28, Xi'an 710049, China
chenpengfei@outlook.com,
{qiy,dihou}@mail.xjtu.edu.cn,
huachong612@gmail.com

Abstract. To provide a high performance and reliable big data platform, this paper proposes a comprehensive invariant-based performance diagnosis approach named *InvarNet-X*. *InvarNet-X* not only covers performance anomaly detection but also root cause inference, both of which are conducted under the consideration of operation context of big data applications. The performance anomaly detection procedure is adopted to trigger the cause inference procedure and accomplished by checking the ARIMA model drift on Cycle Per Instruction (CPI) data of big data applications. The oracle of cause inference is the unobservable root causes of performance problems always expose themselves via the violations of the associations amongst directly observable performance metrics. In *InvarNet-X*, such observable associations as the likely invariants are established by the Maximal Information Criteria (MIC) and each performance problem is signified by a set of violations of those likely invariants. Finally, the root cause is uncovered by searching a similar signature in the signature database. With such a comprehensive analysis, *InvarNet-X* can provide much detailed clues for performance problems and even pinpoint the root causes if the signature database is given. Through experimental evaluations in a small prototype, we find out *InvarNet-X* can achieve an average 91 % precision and 87 % recall in diagnosing some real faults reported in software bug repositories, which is superior to several state-of-the-art approaches. Meanwhile, the local modeling methodology makes *InvarNet-X* easily facilitated in real-time and large scale big data platforms.

Keywords: Big data · Hadoop · Observable likely invariant · Performance diagnosis

1 Introduction

Big data becomes an inevitable trend at present and in the foreseeable future. The popularity of big data attracts many researchers and engineers to devote

J. Zhan et al. (Eds.): BPOE 2014, LNCS 8807, pp. 124–140, 2014.
DOI: 10.1007/978-3-319-13021-7_10

themselves to mining the valuable knowledge in the scrambled data piles. However, during the transformation from 'big data' to 'big value', the performance and reliability of the big data platform deserves the same attention. As a general case, the big data platforms, most if not all, are deployed in large scale distributed systems with thousands of machines using parallel programming such as MapReduce as their program paradigm. In such a huge platform, performance anomalies, faults and failures become commonplace due to the complex interactions in the intricate software stacks [1]. In our previous study [2], we summarized the causes of faults in several widely used open software systems such as Hadoop. One part of the causes are the operational environment changes such as resource utilization hog, workload fluctuation and misconfiguration and the other part are the bugs rooted in the software stacks such as memory leak and lock race. In the big data software stacks, hadoop, no-sql databases, et al. are all the candidate hotbeds of these faults. In addition to that, new faults emerge in the big data platform due to the inherent complexity and three "V"s (i.e. Velocity, Volume and Variety[1]) of big data. The typical bugs are out of memory (OOM) and disk space exhaustion. For instance a bug, MapReduce-1182, shows OOM when the data becomes huge. The bug tells us under the low data-intensive workloads, "shuffle" in memory may be all right but under high data-intensive workloads, the memory is bloated. The faults and failures mentioned above abate the profit brought by big data technology. Thus both of performance and reliability should be the important concerns when setting up a big data platform.

Performance diagnosis as the first line of defending software faults is in charge of finding out the hidden root causes of performance problems. However due to the huge cardinality of suspicious cause set, precise diagnosis in large distributed system is an extraordinarily difficult target to achieve. The difficulty is exacerbated in big data platform embodied in the following aspects.

a. Unlike the web-based applications, the execution duration of big data application is long ranging from several hours to several days (e.g. human genome analysis). Therefore the commonly used QoS metrics like response time or throughput are not suitable any more to monitor in real time. A new key performance indicator (KPI) is urgently needed.
b. The type of big data application varies a lot including both of the batch type and interactive type workloads. These two types of workloads exhibit completely different characteristics and need distinct considerations.
c. The big data platform always possesses tens of thousands of heterogeneous machines which requires the performance diagnosis approach can flexibly adapt to the scale and heterogeneity.

A wide spectrum of research has been done in this field. But most of them focus on fault location in a coarse granularity (e.g. VM or node level [3–5]). Few of them emphasize the root cause inference in a fine granularity (e.g. metric level). Recently an invariant-based performance diagnosis approach is proposed

[1] New properties like "Veracity" are added recently. But we still use the widely accepted three "V"s.

in [6,7] which shares a similar idea with ours. It constructs an invariant network by capturing the stable temporal and spatial relationships amongst the performance metrics collected from the whole distributed system in a pair-wise manner. This approach can work in real time and infer the root causes at fine granularity. However it's insufficient due to its workload agnostic, linear modeling and computationally intractable global invariant construction.

Considering the aforementioned challenges and the weakness of the current research, we propose a comprehensive invariant based performance diagnosis approach, *InvarNet-X*. The **goal** of *InvarNet-X* is to pinpoint the root causes for those problems whose causes are recurrent and investigated[2] and provide some hints for the unknown problems on the fly. To reduce the cost of unnecessary performance diagnosis, *InvarNet-X* first conducts the anomaly detection procedure by checking the autoregressive integrated moving average (ARIMA) model drift on CPI data of big data applications then triggers cause inference procedure. In *InvarNet-X*, each performance problem is signified by a set of violations of likely invariants constructed by MIC [10] and stored in a signature database. Finally, the real culprits are captured by searching the similar signatures in the signature database. *InvarNet-X* works under the consideration of operation context in order to adapt to the varying workloads and hardware heterogeneity. Via experimental evaluations in a small prototype, we find out *InvarNet-X* can achieve an average 91 % precision and 87 % recall in diagnosing some real faults which is superior to several state-of-the-art approaches. Our contribution is three-fold:

- We propose a new performance anomaly detection method by checking ARIMA model drift on CPI data for big data applications.
- We introduce a novel invariant construction method with MIC and build a signature database for each performance problem using the *MIC* invariant.
- We design and implement *InvarNet-X* to evaluate the accuracy and efficacy of our approach. The experimental results show that our approach can find out the culprits accurately.

The rest of this paper is organized as follows. Section 2 depicts the basic idea and problem formulation of *InvarNet-X*. Section 3 demonstrates the details of *InvarNet-X*. Section 4 shows the experimental evaluation and comparisons with several state-of-the-art approaches. Section 5 shows the related work. And Sect. 6 concludes this paper.

2 Problem Formulation

Our work is motivated by the methodology in medical science. The diseases have distinct behaviors from the perspective of some observable symptoms. Thus a conventional method to diagnose a disease is to look for a similar characteristic of observable symptoms from historical knowledge of investigated diseases. The historical knowledge is organized as a 'symptom-disease' database.

[2] These problems take up 50 %–90 % in the known performance problems [8].

In the same manner, the software system can exhibit the distinct behaviors from the viewpoint of performance metrics. The unobservable root causes of performance problems can be investigated via the directly observable runtime performance metrics. The essential work is to build the mapping function from the characteristics of performance metrics to hidden root causes. An ideal function is a one-to-one mapping. As a new exploration, Jiang [6,7] proposed an invariant-based mapping which means the performance states are characterized by the space spanned by the invariants. Our approach shares the similar idea with Jiang's work but makes some improvements. The invariants in this paper are the statistically invariant associations between performance metrics, defined as "observable likely invariant", rather than invariant statements or variables which are stated in previous study [9]. For instance, if the correlation coefficient between "used memory" and "CPU utilization" stays constant, we say these two metrics forms an invariant. Let H denote the monitoring data collected from normal period and F denote the monitoring data from the same system during a recent performance problem (e.g. system hang). Both of H and F comprise n performance metrics: $(M_1, M_2, \cdots, M_n)$. We construct all the invariants of H in a pair-wise manner and make them as the baseline of metric associations. These invariants are denoted by matrix I where each entry I_{M_i,M_j} $(i \neq j)$ represents an invariant formed by metric M_i and M_j. Next, we use the same method to calculate the metric associations of F denoted by matrix A, where each entry A_{M_i,M_j} denotes the association between metric M_i and M_j. If $|I_{M_i,M_j} - A_{M_i,M_j}| \geq \varepsilon$ a violation occurs where ε is the preset threshold, say $\varepsilon = 0.2$ in this paper. All the violations constitute a binary tuple $(0, 1, 1, 0, \cdots, 0)$ ("0" implies no violation,"1" implies violation) which is used to signify a performance problem uniquely. The length of the tuple is determined by the number of entries in matrix I. Aggregating all the binary tuples constructed for multiple performance problems, a signature database is established and will be used in the future performance diagnosis. Different from Jiang's work, we adopt MIC to calculate the metric associations instead of "ARX" [6,7] due to the excellent association discovery power of MIC.

As we know, performance diagnosis is laborious and time-consuming. Hence choosing the right time to conduct performance diagnosis can reduce some unnecessary cost. In our previous work [11], we use the ARIMA model drift on several performance metrics (e.g. CPU utilization) to detect the performance anomaly. However, that method shows weak power to resist the system noise such as the resource utilization fluctuation. Therefore we set up the ARIMA model on CPI instead of other performance metrics. If a performance anomaly is detected, the cause inference procedure is triggered. We first calculate the violation tuple under the current abnormal situation then retrieve a similar signature in the signature database. If a similar signature is found, the culprit is pinpointed otherwise we provide some hints and leave the problem to the system administrators who will manually investigate the problems. Once the performance problems is resolved, a new signature will be added into the signature base.

To adapt to the varying workloads and heterogeneous hardware, we propose the concept of "*operation context*". The operation context contains the workload type and node ID in this paper. *InvarNet-X* works under the consideration of operation context which means the performance model and signature database are built for each workload on each node.

Restrictions: In this paper we only validate our approach in Hadoop-based big data platforms due to its open source and widespread use. When a batch job submitted to Hadoop, Hadoop works in the FIFO mode which means the job takes up the cluster exclusively [12]. This makes *InvarNet-X* distinguish the jobs clearly. But the restriction doesn't exist when Hadoop processes interactive jobs. The performance problem is restricted on performance degradation rather than sudden crash in order to guarantee *InvarNet-X* can collect enough data to proceed diagnosis. From our previous work [2], we observe that large number of bugs can cause performance degradation such as memory leak bug. And Tan [13] also claimed 31 % bug manifested as degraded performance problem in Hadoop. Therefore our system focuses on diagnosing these problems.

Figure 1 demonstrates the basic idea of the this paper. From the figure, we can see the invariant associations between $M_1 - M_2$ and $M_2 - M_3$ on slave-3 are violated. By searching a similar signature in the signature database, we find out the root cause is a CPU-hog.

3 System Design

We adopt a centralized mode to implement *InvarNet-X*. Figure 3 shows the architecture of *InvarNet-X*. *InvarNet-X* leverages the performance metrics and CPI data collected from the Hadoop nodes to build the performance model, invariants

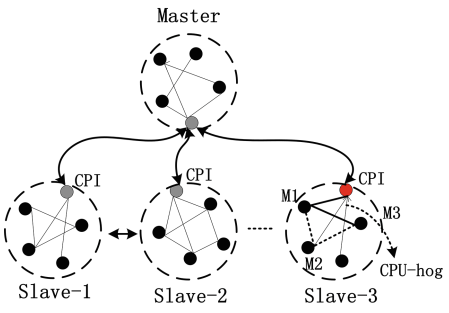

Fig. 1. The basic idea of *InvarNet-X*. Each small circle denotes a performance metric. The line between two performance metrics denotes the invariants and the dash line denotes the violated invariants.

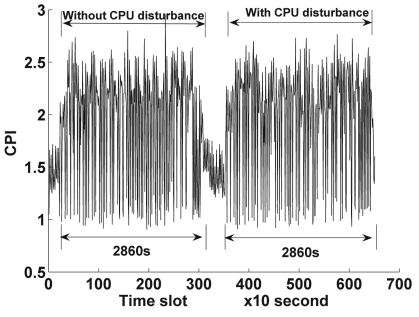

Fig. 2. The CPI and execution time changes of *Wordcount* before and after CPU utilization disturbance. The CPU disturbance starts at 450 point and ends at 480 point

and problem signature database for each batch job and interactive job separately. The output of *InvarNet-X* is a list of root causes which puts the most probable causes in the top. The fault injection module is used to inject faults in Hadoop JobTracker, configuration files, data blocks or operating system in order to validate the effectiveness of *InvarNet-X*. In the following, we will discuss the details of *InvarNet-X*.

InvarNet-X mainly contains two parts and five modules shown in Fig. 3. The offline part contains three modules: performance model building, invariant construction and signature base building. The performance model building module establishes ARIMA models for specific types of workloads to describe the dynamics of CPI data. If the model on CPI data drifts, an anomaly occurs. The invariant construction module is responsible in discovering the MIC invariants amongst the performance metrics. Next, the MIC invariants are fed into the signature base building module which will find out all the violations of invariants under specific performance problems and store the violation tuples as the signatures of corresponding performance problems in a signature database. The online part contains two modules: performance anomaly detection and cause inference. When an anomaly of CPI is detected in performance detection module, the cause inference is triggered. Firstly, a violation tuple is generated by checking all the violations of invariants when the performance problem occurs. Secondly, the signatures in the signature database with a high similarity score to the violation tuple are selected. Finally the root causes corresponding to the selected signatures are reported. Compared to our previous work [11], we make several improvements on two modules including performance anomaly detection and invariant construction, other modules keep the same as before. Before we discuss the details of improvements, we first demonstrate that CPI can be a KPI of big data applications in order to detect the performance anomaly in real time.

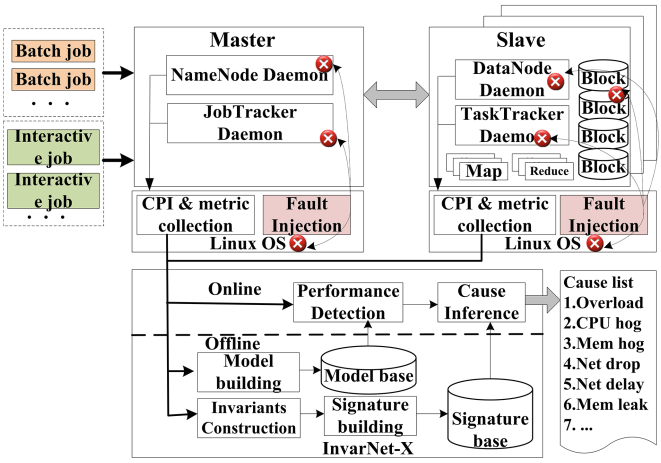

Fig. 3. The architecture of *InvarNet-X*

3.1 CPI as a KPI

In our preliminary work [11], we utilized a specific resource (e.g. CPU or memory) utilization as the KPI. It indeed indicates some performance problems in most cases. But it may mislead anomaly detection result under the disturbance of system noise. To validate this perspective, we inject resource utilization disturbance to mimic the system noise when the job (e.g. *Wordcount*) is running. From the results we observe that the execution time of some jobs have no changes although they are suffering from anomalies. Figure 2 shows the CPI changes of *Wordcount* before and after the CPU utilization disturbance (additional 30 % CPU utilization for 300 s). The CPU disturbance doesn't enlarge the execution time while the CPI keeps unaffected. Therefore a more robust KPI should be proposed to reflect the performance of the big data application.

For a specific program compiled to run on a specific machine, the execution time of this program could be expressed as:

$$T = I * CPI * C,$$

where I denotes the total instructions of this program, CPI denotes cycles per instruction, C denotes the time length (second) of one cycle. In this equation, I and C are fixed. Hence the execution time T only depends on CPI and CPI can be a candidate KPI of the big data application. To further validate this new KPI, we choose several types of jobs in BigDataBench [14] including batch type: *Wordcount, Sort, Bayes classifier* and interactive type: *TPC-DS* workloads (8 queries run in a mixed mode). 15 GB test data is generated using the BigDataBench. Four-group tests are designed. In each group, only one type of job is repeated for 25 times. And during the job running, we inject several faults such as network jam, CPU hog and disk hog to make the execution time of these jobs varies. During each time of running we collect the CPI data every 10 s and employ the

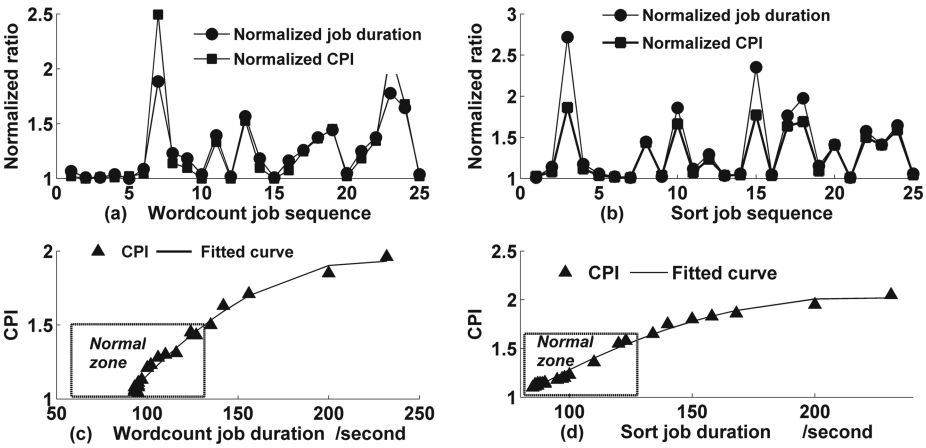

Fig. 4. The CPI changes with the execution time of Hadoop jobs

95 % percentile of CPI data as a sufficient statistics for one run. Other statistics like average are also applicable. For each job, the execution time and the 95 % percentile of CPI data is normalized to the minimum value respectively in one group. Due to the limited space, we only show the CPI data and execution time of *wordcount* and *sort* in Fig. 4. From Fig. 4 (a) and (b), we observe that the CPI changes with the execution time consistently. The correlation coefficient of these two metrics is 0.97 and 0.95 for *wordcount* and *sort* respectively. Figure 4 (c) and (b) demonstrate the scatter plot of CPI and execution time. And we use a 2-order polynomial function to fit these data and conclude that CPI increases monotonously with the job execution time. All the evidences show that CPI can be a stable performance indicator of big data applications. Actually, Zhang et al. [16] also utilize CPI as a performance indicator of CPU interference.

3.2 Performance Anomaly Detection

We employ the method proposed in our previous work [11] to detect the performance anomaly. But this paper uses CPI data rather than conventional performance metrics to build ARIMA model. If the reader wants to know the detail of ARIMA model building, please refer to [11]. The ARIMA model of CPI data in the normal state is first established and stored in an XML file in a five-tuple: *(p, d, q, ip, type)* format where the first three elements are the parameters of ARIMA, *ip* is the ip address of a Hadoop node and *type* is the workload type. To model the distinct characteristics of CPI data at "Map" and "Reduce" phases of Hadoop workloads, we utilize N (e.g. 10) complete normal execution traces of CPI data of a specific type of workload to train the ARIMA model. When a new job arrives at the platform, *InvarNet-X* selects a performance model for performance anomaly detection from the archived models instantly. A simple threshold based anomaly detection method is proposed in [11]. That is if

$$\xi = |M'_{cpi}(t) - M_{cpi}(t)| > \alpha$$

a performance anomaly occurs where $M_{cpi}(t)$ is the CPI data at time t, $M'_{cpi}(t)$ is the CPI data predicted by ARIMA model using previous CPI data and α is the preset threshold. But how to set the threshold still remains a problem. In this paper, we propose three guiding rules to set the threshold. Each type of workload is repeated for N times, say 20 under normal state. Next, we use the trained ARIMA model to fit the CPI data during N runs. The absolute value of fitting residual is denoted by R. The three rules are listed below. To make the performance anomaly detection more robust to resist system noises, we report a performance problem when the anomaly occurs for three times continuously. The effectiveness of these rules will be discussed in the Sect. 4.

- **max-min**. Use $max(R)$ as the upper bar, $min(R)$ as the lower bar. If $\xi > max(R)$ or $\xi < min(R)$, an anomaly occurs.
- **95-percentile**. Use the 95 % percentile of R as the threshold.
- **beta-max**. Use $\beta * max(R)$ as the threshold where β is a fluctuation factor which is used to cover the unobserved value escaped from the test. We set $\beta = 1.2$ in this paper.

3.3 Invariants Construction

We use MIC to discover the association between two performance metrics. The detailed description of MIC could be found in [10]. Fore each metric pair X, Y, their association coefficient is represented by the $MIC(X, Y)$ score which falls in the region $[0, 1]$. In this paper, a simple but exhaustive pair-wise search method is adopted to calculate all the associations. Suppose M metrics are collected from a specific node, in theory, $M(M-1)/2$ association pairs should be generated. However not all of the association pairs are invariants. The stable ones which don't fluctuate too much under the normal state are regarded as the invariants. Under the normal state, one type of workload is repeated for N times. We use an association matrix to save the association pairs, denoted as A^i where the superscript denotes the ith run and each entry $A^i(m, n)$ denotes the MIC score of metric m and metric n. Let the vector $V(m, n) = (A^1(m, n), A^2(m, n), \cdots, A^N(m, n))$ denote the association coefficients of metric pair (m, n) over N runs. If a association pair doesn't exist in one run, the MIC score is assigned 0. We further select the association pairs satisfying the following condition: $Max(V(m, n)) - Min(V(m, n)) < \tau$. The threshold τ is a tunable parameter and is set 0.2 in this paper. The invariant selection algorithm is shown in Algorithm 1. When all the invariants for one type of workload are discovered, we store them in an XML file as a three-tuple $(I, ip, type)$ where I stores all invariants in a matrix format, ip is the ip address of a Hadoop node and $type$ is the workload type.

Algorithm 1. Invariant selection

Input: A set of performance metrics in the N runs under the same workload in the normal state: $P^1 = (P_1^1, P_2^1, \cdots, P_M^1)$, $P^2 = (P_1^2, P_2^2, \cdots, P_M^2)$, $\cdots$, $P^N = (P_1^N, P_2^N, \cdots, P_M^N)$, M is the number of performance metrics;
Input: A preset threshold τ
Output: The set of invariants I;

```
for i = 1; i = N; i++ do
  for each metric m ∈ P^i do
    for each metric n ∈ P^i do
      A^i(m, n) = MIC(m, n);
    end for
  end for
end for
for each metric m ∈ P^1 do
  for each metric n ∈ P^1 do
    for i = 1; i = N; i++ do
      V(m, n) ← A^i(m, n) // V(m, n) is a vector
    end for
    if Max(V(m, n)) − Min(V(m, n)) < τ then
      I(m, n) ← Max(V(m, n))
    end if
  end for
end for
```

For each performance problem whose root cause is investigated (e.g. memory leak), we build the association matrix $A_{abnormal}$ when the performance problem occurs. Then we compare $A_{abnormal}$ with the invariants I of the same workload in the same Hadoop node and find out all the violations according to Sect. 2. The violations constitute a binary tuple and the tuple acts as the signature of one performance problem. The signature is stored in the signature database in the four-tuple format: *(binary tuple, problem name, ip, workload type)*. As more performance problems are diagnosed, the number of items in signature database increases gradually.

If the cause inference procedure is triggered, we adopt the approach mentioned in [11] to report the most probable root cause whose similarity score is the most close to the violation tuple. Until now the performance diagnosis is finished.

4 Experimental Evaluation

We have implemented a prototype and deployed it in a controlled environment. To collect the process and operating system performance metrics, a low overhead and off-the-shelf tool, *collectl*, is employed. The collected 26 performance 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. "perf" tool is used to collect the cycle and instruction periodically by reading the corresponding registers in the hardware performance counter on a per process basis. The collection interval is 10 s. Other parts of *InarNet-X* are developed from scratch. In the following, we will give the details of our experimental methodology and evaluation results.

4.1 Evaluation Methodology

Due to the lack of real operating platforms, our approach is only evaluated in a controlled big data platform. But we believe it works well in a real system without exceptions. The controlled platform contains five server machines hosting the benchmark. Each physical machine is configured with two 4-core Xeon 2.1 GHZ CPU processors, 16 GB memory, a 1 TB hard disk and a gigabit NIC and runs a 64-bit CentOS 6.2. All the servers are interconnected by a 8-port gigabit Switch. We adopt Hadoop 1.0.2, Mahout 0.6, Hive 0.9 and Mysql 5.1 as the primary software stack.

In this paper we choose four batch type of workloads: *Sort, Wordcount, Grep and Naive Bayesian classifier* and one interactive type of workloads: *TPC-DS* in BigDataBench, leaving other workloads for the future work. And the 8 queries in *TPC-DS* run simultaneously in a mixed mode. During all the experiments, 15 GB data is generated by the tool in BigDataBench benchmark. According to the reports in previous literature [13] and Hadoop bug repository [15], we inject the following faults. For the performance problems caused by runtime environment changes, we inject the following faults: (1) CPU-hog: a CPU-bound

application co-locates with TaskTracker competing for CPU resource sharply; (2) Mem-hog: a memory-bound application consumes a large number of memory on one data node; (3) Disk-hog: we use a disk-bound program to generate a mass of disk reads and writes on the data node; (4) Net-drop: we use a fault injection tool "AnarchyApe" to mimic the packet loss on the name node; (5) Net-delay: we use "AnarchyApe" to delay all the packets for 800 ms; (6) Block Corruption (Block-C): we use "AnarchyApe" again to corrupt some data blocks on one data node; (7) Misconf: we set a low value (e.g. 1M) for the item "mapred.max.split.size" in the configure file; (8) Overload: we increase the current number of interactive type of workloads; (9) Suspend: we use "AnarchyApe" again to suspend the datanode or tasktracker process. For the performance problems caused by software bugs, we inject the following faults: (1) RPC-hang: the bug HADOOP-6498 causes rpc call hang. To reproduce this bug, we use hadoop inject framework to add a "sleep" statement to delay RPC call; (2) HADOOP-9703 (H-7703): when the method "stop" of "org.apache.hadoop.ipc.Client" is invoked, the thread leak happens. We use the hadoop fault inject framework to reproduce the bug by invoking this function call. (3) HADOOP-1036 (H-1036): we revert Hadoop to an older version and trigger the bug by throwing NullPointerException; (4) Lock-R: we use hadoop fault inject framework to substitute the method who has the property "synchronized" with a new method without "synchronized"; (5) HADOOP-1970 (H-1970): hadoop fault inject framework is also used to trigger this bug by interfering the communication thread; (6) Block receiver exception (Block-R): we add an exception statement in the "receivePacket" function of Class BlockReciever by hadoop inject framework. All the injected faults are guaranteed to cause significant performance problems.

Each fault mentioned above is repeated for 40 times and lasts 5 min. Two of them are used to train the signatures and the others are used to cause inference. As the probability of multiple faults happening in the same node at the same time is very tiny, we don't consider multiple faults in this paper. Actually, our method could be easily extended to multiple faults by listing multiple root causes whose signatures are most similar to the violation tuple. We leverage two commonly used metrics: precision and recall to evaluate the effectiveness of our prototype.

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

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

4.2 Performance Anomaly Detection

We use the performance anomaly detection method proposed in Sect. 3.2 to detect the anomalies incurred by fault injections. Figure 5 shows the CPI prediction residuals of *Wordcount* and *TPC-DS* using the trained ARIMA before and after CPU-hog injection. Even a cursory glance at this figure, we can see the

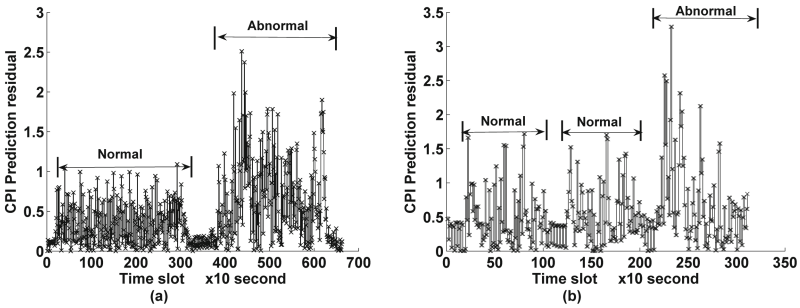

Fig. 5. The CPI prediction residuals before and after CPU-hog injection. (a) shows the CPI prediction residuals of workload *Wordcount*; (b) shows the CPI prediction residuals of workload *TPC-DS*

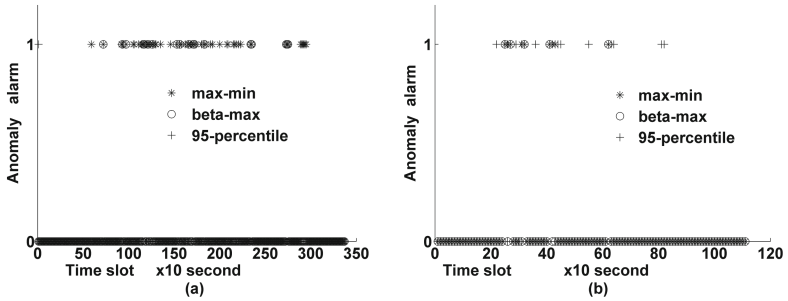

Fig. 6. The anomaly detection results of "max-min", "95-percentile" and "beta-max". (a) shows the results under workload *Wordcount*; (b) shows the results under workload *TPC-DS*

anomaly occurs when the CPU-hog is injected. We use the normal CPI data to train ARIMA model and use the CPI data with CPU-hog to detect anomalies. The result of anomaly detection is shown in Fig. 6 where "1" on y-axis denotes anomaly. According to the ground truth, we observe that the 95 %-percentile method has the worst detection result while the other two methods have very similar results. However the "max-min" method has a larger computational complexity than "beta-max" method due to additional "min" operation. Hence we choose "beta-max" method as the final performance anomaly detection method.

4.3 Diagnosis Results

We evaluate *InvarNet-X* under both of batch type of workloads and interactive type of workloads. Due to the limited space, we only show diagnosis results under workload *Wordcount* and *TPC-DS*. In reality, the diagnosis results under other workloads such as *Sort* are very similar to the shown results. Figure 7 shows the diagnosis result under workload *TPC-DS*. From this figure, we observe that

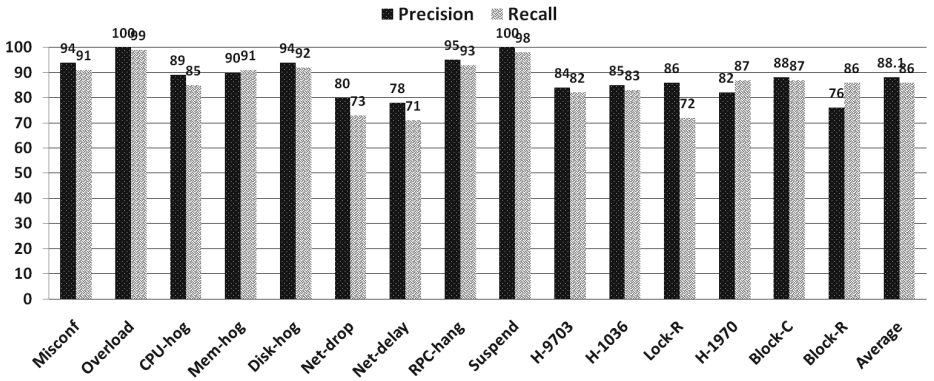

Fig. 7. The diagnosis result under workload *TPC-DS*

InvarNet-X achieves the perfect precision (100 %) and recall (99 %, 98 %) for *Overload* and *Suspend*. Because these two faults can cause a large number of violations of invariants which makes them easily distinguished from other faults by *InvarNet-X*. However the recall of *Lock-R* is very low as *Lock-R* makes different violations in different runs leading to a high false positive. For these non-deterministic problems, although *InvarNet-X* can't precisely pinpoint the root causes, it can provide some hints by showing the violated association pairs (e.g. "lock number-cpu utilization") Another interesting finding is the low accuracy of *Net-drop* and *Net-delay*. Comparing the diagnosis results with the ground truth, we find *InvarNet-X* mistakes *Net-drop* for *Net-delay* and vise versa sometimes because these two faults have very similar signatures. That's a typical "signature conflict" which will be discussed in our future work. Figure 8 shows the diagnosis result under workload *Wordcount*. When Hadoop works in FIFO mode, one job takes up the whole cluster exclusively. Therefore *overload* doesn't happen in this situation. Besides some similar characteristics with *TPC-DS*, the average precision (91.2 %) and recall (87.3 %) of *Wordcount* are higher than the average precision (88.1 %) and recall (86 %) of *TPC-DS*. That's because *TPC-DS* is a mixed workload including multiple different queries which may skew the performance model (i.e. ARIMA) and invariants even in the normal state. While *Wordcount* as a single batch job keeps a stable performance model and invariants in the normal state. In other words, the batch type of workloads possess higher quality of signatures.

Similar to our work, Jiang et al. [6,7] also propose an invariant based performance diagnosis approach. In their work, they use autoregressive models with exogenous inputs (ARX) to learn linear relationships between performance metrics. To compare with their work, we use ARX instead of MIC to implement the invariant construction. And to further validate the necessity of operation context, we implement another version of *InvarNet-X* without operation context which only contains a single performance model and signature base for one specific workload. Due to the limited space, we only show the diagnosis results

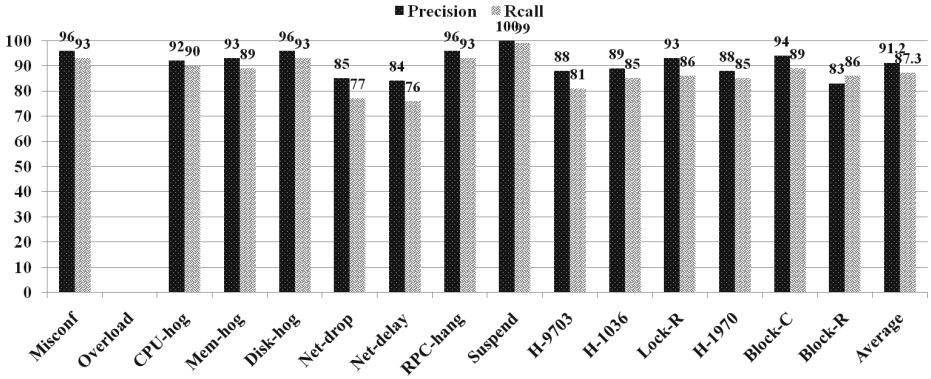

Fig. 8. The diagnosis result under workload *Wordcount*

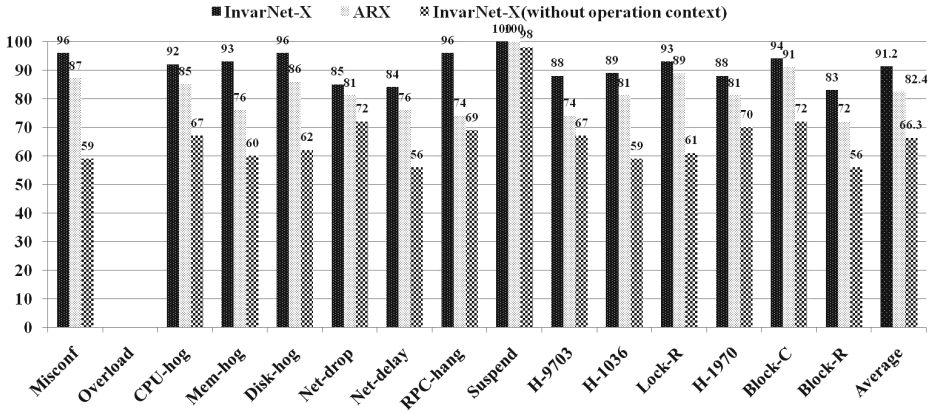

Fig. 9. The comparison of *InvarNet-X*, *ARX* and *InvarNet-X* (no operation context) in precision

under workload *Wordcount* in Figs. 9 and 10. Figure 9 and Fig. 10 show the diagnosis precision and recall of *InvarNet-X*, *ARX* and *InvarNet-X* (no operation context) respectively. From these two figures, we observe that the diagnosis precision of *InvarNet-X* is about 9 % higher than the one of *ARX* while the diagnosis recall shows no significant differences. The invariants discovered by *ARX* are rigorous linear relationships. The linear relationships can be broken easily when a performance problem occurs meaning that *ARX* has a strong power to capture the performance problems. However it has a weak power to distinguish the performance problems due to many similar signatures. This is the reason why we obtain the above observation. *InvarNet-X* without operation context shows a very disappointing diagnosis accuracy no matter in precision and recall. Therefore operation context is a necessary factor of performance diagnosis.

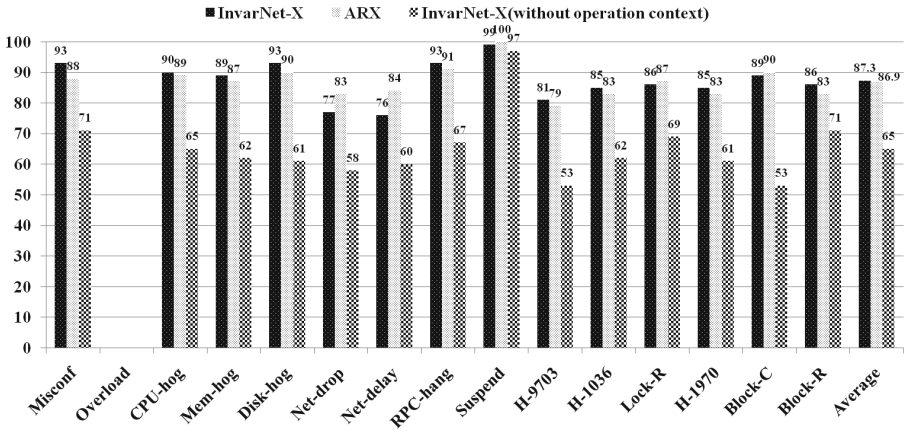

Fig. 10. The comparison of *InvarNet-X*, *ARX* and *InvarNet-X* (no operation context) in recall

Table 1. The overhead of *InvarNet-X* and *ARX* (/second)

Workload	Perf-M	Invar-C	Invar-C (*ARX*)	Sig-B	Perf-D	Cause-I	Cause-I (*ARX*)
Wordcount	1.2	45	700	4	0.02	1.6	10
Sort	0.8	30	650	3	0.03	1.7	11
Grep	0.2	18	410	1.7	0.02	1.6	10
Interactive	0.5	16	380	1.5	0.03	1.6	12

4.4 Overhead

Here we only consider the CPU overhead because other types of overhead like memory and disk caused by *InvarNet-X* are very small. The CPU overhead contains six parts: data collection, performance model building (Perf-M), invariant construction (Invar-C), signature building (Sig-B), performance anomaly detection (Perf-D) and cause inference (Cause-I). The data collection costs no more than 5 % CPU utilization. Table 1 shows the execution time of the other five parts under different types of workloads. We observe that the execution time of Perf-D and Cause-I stays below 2 s satisfying the online requirement. While the execution time of Cause-I(*ARX*) is around 10 s much larger than Cause-I(*InvarNet-X*). Although as an offline part, the execution time Invar-C (*InvarNet-X*) is up to 45 s, it is much lower than the one of Invar-C (*ARX*) in one order of magnitude. Therefore *InvarNet-X* is computationally tractable when it scales up in large scale big data platform.

5 Related Work

A large quantity of work has been done in performance diagnosis of general distributed systems. However most of them are concerned with fault location in a coarse granularity (e.g. VM level [3–5]). Few of them emphasize the root cause inference in a fine granularity. Towards performance diagnosis in MapReduce (e.g. hadoop) system, the work could be roughly categorized into two classes: log-based and correlation-based. The log-based method [13] can pinpoint the buggy code in a very fine granularity (e.g. line of code) but it is hard to conduct in real time. The correlation-based method [5] more often than not uses the *peer-similarity* to find out the abnormal nodes assuming that the correlations amongst the performance metrics of different nodes are stable. However an exceptional case exists. Assume one bug exists in the platform, when the bug is triggered by a certain job, say *wordcount*, all the nodes behave abnormally in a similar way but the correlations are not deviated. In this case, the correlation-based method will ignore this fault.

Recently an invariant-based performance diagnosis approach is proposed in [6,7]. It constructs an invariant network by capturing the stable temporal and spatial relationships amongst the performance metrics in a pair-wise manner. A set of deviations of these invariants can indicate a specific fault. This approach can work in real time and infer the root causes at fine granularity. However this method has the following limitations: (a) It is workload agnostic. As pointed in [7], they selected 111 measurements form the system and 74 of them are correlated with the workload. Moreover according to our work [11], it's hard to find out such a model suitable to all kinds of workloads. Hence workload is an important factor in performance diagnosis. (b) It only considers linear relationships between performance metrics leading to invariant missing. Due to highly dynamical nature of software system, non-linearity is a more common case. (c) The global constructions of invariant network and simultaneously checking all the invariants in real time make it computationally intractable in the large scale distributed environment.

6 Conclusion

This paper proposes a comprehensive invariant-based approach, *InvarNet-X*, to pinpoint the culprits of performance problems in the big data platform. *InvarNet-X* not only covers performance anomaly detection but also root cause inference. The performance anomaly procedure is accomplished by checking the ARIMA model drift on CPI data of big data applications. In *InvarNet-X*, the likely invariants are established via MIC and each performance problem is signified by a set of violations of those likely invariants. Finally, the root cause is uncovered by searching a similar signature in the signature database. Through experimental evaluations in a small prototype, we find out *InvarNet-X* can achieve an average 91 % precision and 87 % recall in diagnosing some real faults which is superior to several state-of-the-art approaches. Meanwhile *InvarNet-X* causes a low overhead to the system.

Acknowledgments. We thank to all the members in our research group.

References

1. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
2. Chen, P., Qi, Y., Hou, D., Zheng, P.: CauseInfer: automatic and distributed performance diagnosis with hierarchical causality graph in large distributed systems. In: 33rd Annual IEEE International Conference on Computer Communications, Toronto (2014)
3. Bodik, P., Goldszmidt, M., Fox, A., Woodard, D.B., Andersen, H.: Fingerprinting the datacenter: automated classification of performance crises. In: 5th European Conference on Computer Systems, pp. 111–124. ACM Press, Lancaster (2010)
4. Nguyen, H., Shen, Z., Tan, Y., Gu, X.: FChain: toward black-box online fault localization for cloud systems. In: 33rd International Conference on Distributed Computing Systems (ICDCS), pp. 21–30. IEEE Press, Philadelphia (2013)
5. Kang, H., Chen, H., Jiang, G.: PeerWatch: a fault detection and diagnosis tool for virtualized consolidation systems. In: 7th International Conference on Autonomic Computing, pp. 119–128. ACM Press, London (2010)
6. Jiang, G., Chen, H., Yoshihira, K.: Efficient and scalable algorithms for inferring likely invariants in distributed systems. IEEE Trans. Knowl. Data Eng. **19**(11), 1508–1523 (2007)
7. Jiang, G., Chen, H., Yoshihira, K.: Discovering likely invariants of distributed transaction systems for autonomic system management. In: 3rd IEEE International Conference on Autonomic Computing, pp. 199–208. ACM Press, New York (2006)
8. Duan, S., Babu, S., Munagala, K.: Fa: a system for automating failure diagnosis. In: 25th IEEE International Conference on Data Engineering, pp. 1012–1023. IEEE Press, Shanghai (2009)
9. Ernst, M.D., Perkins, J.H., Guo, P.J., McCamant, S., Pacheco, C., Tschantz, M.S., Xiao, C.: The Daikon system for dynamic detection of likely invariants. Sci. Comput. Program. **69**(1), 35–45 (2007)
10. Reshef, D.N., Reshef, Y.A., Finucane, H.K., Grossman, S.R., McVean, G., Turnbaugh, P.J., Sabeti, P.C.: Detecting novel associations in large data sets. Science **334**(6062), 1518–1524 (2011)
11. Chen, P., Qi, Y., Li, X., Su, L.: An ensemble MIC-based approach for performance diagnosis in big data platform. In: 1st IEEE International Conference on Big Data, pp. 78–85. IEEE Press, Santa Clara (2013)
12. Sangroya, A., Serrano, D., Bouchenak, S.: Benchmarking dependability of MapReduce systems. In: 31st IEEE International Symposium on Reliable Distributed Systems, pp. 21–30. IEEE Press, Irvine (2012)
13. Tan, J., Pan, X., Marinelli, E., Kavulya, S., Gandhi, R., Narasimhan, P.: Kahuna: problem diagnosis for MapReduce-based cloud computing environments. In: 12th IEEE/IFIP Network Operations and Management Symposium, pp. 112–119. IEEE Press, Osaka (2010)
14. Wang, L., Zhan, J., Luo, C., et al.: BigDataBench: a big data benchmark suite from internet services (2014). arXiv preprint arXiv:1401.1406
15. Hadoop bug repository. http://hadoop.apache.org/issue_tracking.html
16. Zhang, X., Tune, E., Hagmann, R., et al.: CPI2: CPU performance isolation for shared compute clusters. In: 8th ACM European Conference on Computer Systems, pp. 379–391. ACM Press, New York (2013)