Ren R, Cheng J, He XW *et al.* HybridTune: Spatio-temporal performance data correlation for performance diagnosis of Big Data systems. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 34(6): 1167–1184 Nov. 2019. DOI 10.1007/s11390-019-1968-y

HybridTune: Spatio-Temporal Performance Data Correlation for Performance Diagnosis of Big Data Systems

Rui Ren[1,2], *Member*, *CCF*, *IEEE*, Jiechao Cheng[3], Xi-Wen He[1], Lei Wang[1], *Member*, *CCF*
Jian-Feng Zhan[1,*], *Member*, *CCF*, *ACM*, *IEEE*, Wan-Ling Gao[1], *Member*, *CCF*, *ACM*, *IEEE*
and Chun-Jie Luo[1,2], *Member*, *CCF*

[1] *Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China*
[2] *University of Chinese Academy of Sciences, Beijing 100049, China*
[3] *School of Computing, National University of Singapore, Singapore 117417, Singapore*

E-mail: renrui@ict.ac.cn; jetrobert19@gmail.com
{hexiwen, wanglei_2011, zhanjianfeng, gaowanling, luochunjie}@ict.ac.cn

Received September 6, 2018; revised September 4, 2019.

Abstract With tremendous growing interests in Big Data, the performance improvement of Big Data systems becomes more and more important. Among many steps, the first one is to analyze and diagnose performance bottlenecks of the Big Data systems. Currently, there are two major solutions. One is the pure data-driven diagnosis approach, which may be very time-consuming; the other is the rule-based analysis method, which usually requires prior knowledge. For Big Data applications like Spark workloads, we observe that the tasks in the same stages normally execute the same or similar codes on each data partition. On basis of the stage similarity and distributed characteristics of Big Data systems, we analyze the behaviors of the Big Data applications in terms of both system and micro-architectural metrics of each stage. Furthermore, for different performance problems, we propose a hybrid approach that combines prior rules and machine learning algorithms to detect performance anomalies, such as straggler tasks, task assignment imbalance, data skew, abnormal nodes and outlier metrics. Following this methodology, we design and implement a lightweight, extensible tool, named HybridTune, and measure the overhead and anomaly detection effectiveness of HybridTune using the BigDataBench benchmarks. Our experiments show that the overhead of HybridTune is only 5%, and the accuracy of outlier detection algorithm reaches up to 93%. Finally, we report several use cases diagnosing Spark and Hadoop workloads using BigDataBench, which demonstrates the potential use of HybridTune.

Keywords Big Data system, spatio-temporal correlation, rule-based diagnosis, machine learning

1 Introduction

Recently, computing industry has witnessed an unprecedented increasing popularity of Big Data. Optimizing the performance of Big Data systems is a big concern in both academia and industry. However, considering the configuration diversity and the system complexity, performance analysis and optimization face great challenges. First, the node configurations, e.g., processors, memory, disks, and networks, are multitudinous among homogeneous or heterogeneous clusters, especially varied accelerator configurations. Second, the software stacks of Big Data systems usually have hundreds of adjustable parameters, and further increase the difficulties of performance optimization.

A series of research efforts focus on the performance analysis and optimization of Big Data systems, such as Hitune[1], SONATA[2], Theia[3], Mochi[4], Artemis[5], and Starfish[6]. Among them, either data-driven or rule-based analysis methods are used in these performance analysis tools. The pure data-driven diagnosis approach is promising for simple distributed applications, while time-consuming and difficult for complex Big Data systems. Meanwhile, the rule-based analysis

Regular Paper
This work is supported by the National Key Research and Development Program of China under Grant No. 2016YFB1000601.
*Corresponding Author

methods usually require prior knowledge, which is difficult to obtain in real scenarios. For example, when the system performance deteriorates, it is difficult to build priori rules online to find the bottleneck immediately.

For Big Data applications like Spark workloads, we observe that the tasks in the same stage normally execute similar or even the same codes on each data partition, which we call stage similarity. Taking advantages of the stage similarity and distributed characteristics of Big Data systems, we analyze the behaviors of the Big Data applications in terms of both system and architecture metrics of each stage, and propose a hybrid approach that utilizes prior knowledge rules or data-driven machine learning algorithms to detect performance anomalies, such as straggler tasks, task assignment imbalance, data skew, abnormal nodes and performance outlier.

For specific performance data, either priori knowledge rules or data-driven machine learning algorithms are used to detect performance anomalies. For instance, prior knowledge rules (e.g., threshold setting) are used to detect straggler tasks, task assignment imbalance and skew data size; while for detecting uneven data placement, abnormal nodes and performance outlier, we use machine learning algorithms, such as the Euclidean distance outlier algorithm for uneven data placement detection, the cosine similarity-based approach for abnormal node detection, and the performance outlier detection approach based on distance and magnitude.

Following this methodology, we design and implement a lightweight, extensible tool, named HybridTune. It collects the performance data of the whole software and hardware stacks without modifying user program and software.

Then we validate the overhead and anomaly detection effectiveness of HybridTune using BigDataBench[7] workloads. Our experiments show that the accuracy of outlier detection reaches 93%, while the overhead caused by HybridTune is lower than 5%. Also, we take several Spark and Hadoop workloads as examples, to demonstrate how HybridTune supports the performance analysis and diagnosis efficiently on Big Data applications.

The rest of the paper is organized as follows. Section 2 states our motivation. Section 3 introduces the spatio-temporal characteristics of Big Data systems. Section 4 describes how to detect performance anomalies in different scenarios. Section 5 presents the HybridTune implementation. Section 6 presents the experiments. Section 7 discusses several case studies. A brief discussion of related work is presented in Section 8. Finally, conclusions are drawn in Section 9.

2 Motivation

Since the distribution characteristics of Big Data systems, the performance problems may exist on a large number of subsystems, such as multi-core processors, memory subsystems, disk, and network subsystems. When some performance bottlenecks occur on a certain subsystem, it is likely to cause some performance problems in other subsystems. This means that the performance issues of different subsystems can be correlated and spread. Moreover, Big Data applications are always built on complex software stacks, and the deeper the software stacks, the more the factors that affect the performance. Fig.1 illustrates the stack of Big Data systems and causes of performance degradation. For instance, the affecting factors of bad performance include: 1) the error user operation, code bugs, data skew or application co-location in the application level; 2) the imbalance scheduling, improper configuration parameters of software framework (e.g., configuration of Hadoop/Spark, Java heap, GC (Garbage Collection)) in software stack level; 3) resource contention, system failure, bad OS parameters (e.g., NUMA (Non Uniform Memory Access Architecture), memory page) in the system level; 4) the hardware configurations (e.g., SMT (Simultaneous Multi-Threading), memory capacity) and machine crashing in the architecture and hardware level.

Taking Hadoop as an example, if the data is partitioned on a key space that has too little entropy, i.e., a few keys correspond to a lot of data, then the partitions will differ in sizes[8], and the corresponding tasks will run longer time because they have a lot of work to do. For system resource utilization level, if the disk loads on one node are much heavier than those of the other nodes, the reduce tasks on the node that has heavy disk loads are likely to be outliers. For runtime level, the behavior of a MapReduce job in Hadoop is controlled by the settings of more than 190 configuration parameters[6], such as HDFS configuration, and task scheduling parameters. The impacts of so many parameters on performance are intricate. In addition, the performance issues of different levels may be interrelated and cross-correlated.

Thus, in order to find out a variety of anomalies that exist at different levels, we need to collect multiple levels of performance data, and use specific data

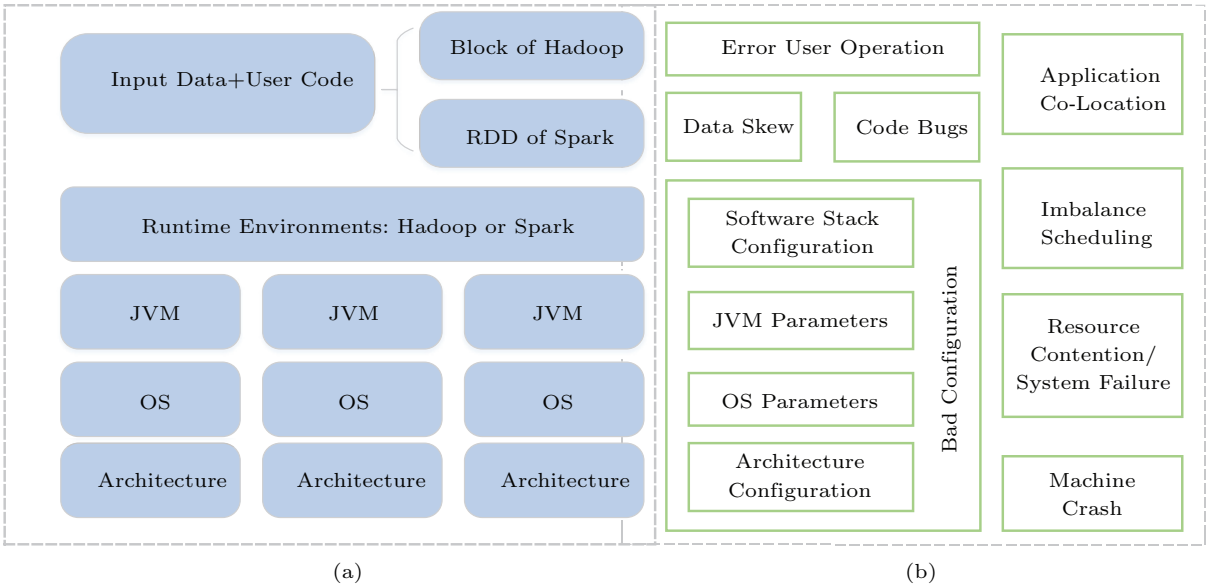


Fig.1. (a) Stack of Big Data systems and (b) causes of performance degradation.

types (e.g., configuration parameters, data sizes, task execution status, and resource usages) and abnormal determination rules for fine-grained anomaly detection. Simultaneously, to find out whether the performance issues at different levels have mutual influence or correlation relationship, we need to conduct unified association analysis for the multiple performance data.

3 Spatio-Temporal Data Correlation

Generally speaking, jobs on Big Data systems are divided into several stages, and each stage owns multiple tasks, which are assigned to multiple distributed nodes. That means Big Data applications have the temporal-spatial characteristics.

1) *Stage Characteristics.* We observe that the jobs on different Big Data systems are generally executed in stages or in the stage-like process, and tasks in the same stage are normally executed by the same or similar code on the data partition. In other words, the Big Data applications have the characteristics of stage similarity. For example, a Hadoop job executes in two stages: Map and Reduce. In the map phase, several map tasks are executed in parallel to process the corresponding input data. After all map tasks are finished, the intermediate results are transferred to the reduce tasks for further processing. Spark splits stages and partition data, and then generates the DAG (Directed Acyclic Graph) of stage dependency specifically, and the tasks in one stage are executed by the same code and portioned into various data fragments[9]. A Dryad job contains a set of stages and each stage consists of an arbitrary set of vertex (each vertex runs on a distinct data partition) replicas, and meanwhile all graph edges of the stage constitute point-to-point communication channels[10].

2) *Distribution Characteristics*[11]. As Big Data systems are used for processing a huge amount of data, a single node is impossible to complete the big tasks quickly. Therefore Big Data systems generally use the large-scale distributed cluster architecture. They mainly utilize the distributed file system or distributed database to store data, and use parallel programming model to process tasks. That is to say, the tasks of Big Data applications will be scheduled into different nodes. And the distributed parallel architect distributes data across multiple servers, and it can improve data processing speeds. Therefore we consider that the Big Data applications have distribution characteristics in spatial dimension.

Based on these characteristics, we propose a spatio-temporal correlation approach that involves timestamp information and distributed nodes information to process the data in multiple layers, which is shown in Fig.2. Specifically, in order to implement association in temporal dimension, the time synchronization of clusters must be guaranteed. Here, the network time protocol (NTP) is used to synchronize computer time in our Big Data systems. Then, according to the stage characteristics, we combine the stage runtime information of Big Data applications with the resources utilization.

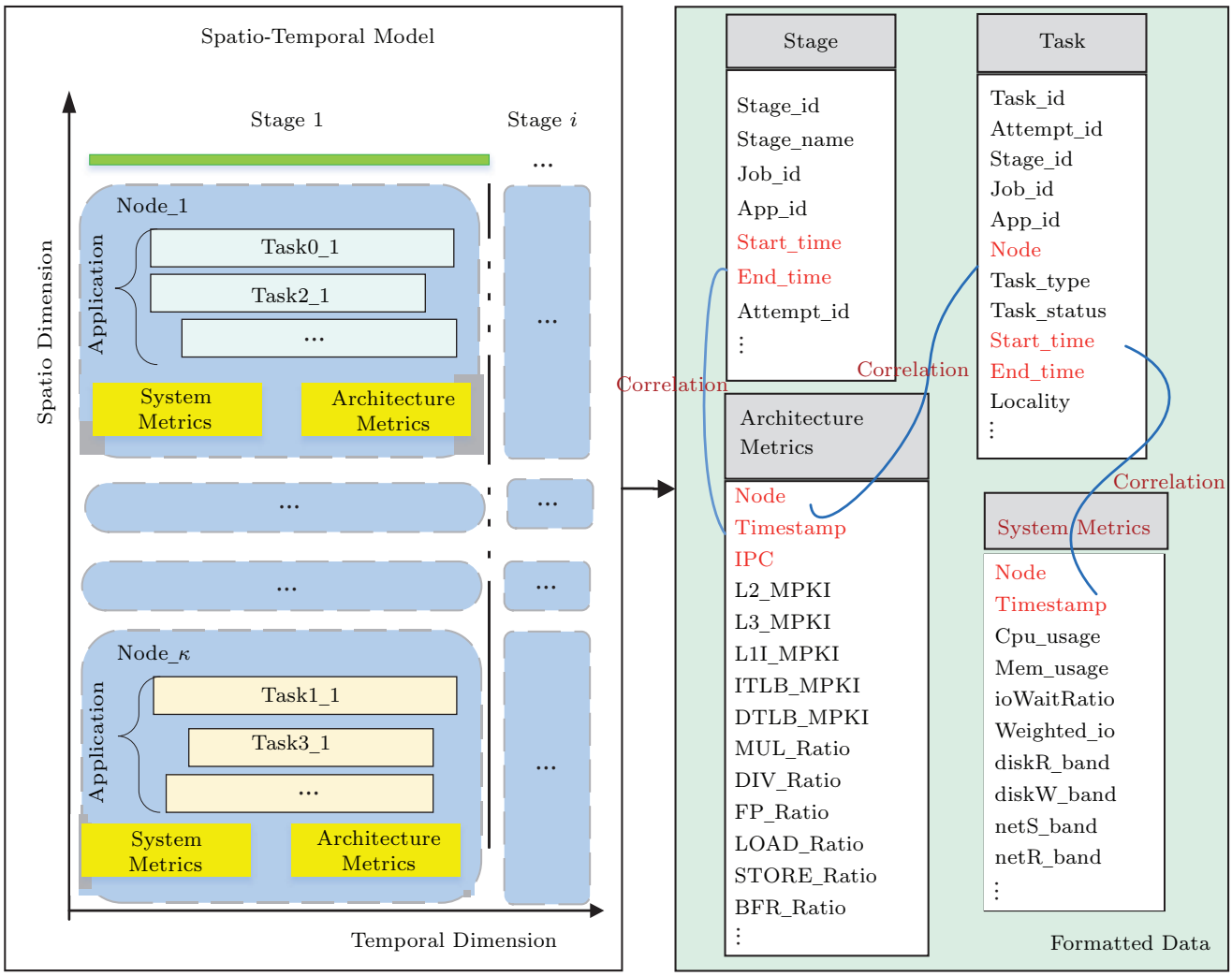


Fig.2. The collected multi-layer performance data is processed into formatted data with timestamp information and nodes information, and correlated based on the spatio-temporal characteristics of Big Data systems.

4 Anomalies Detection

After the multi-level performance data is correlated through the spatio-temporal correlation method, we propose several anomalies detection approaches for straggler tasks, task assignment imbalance, data skew, abnormal node and outlier metrics. For different types of anomalies, we first select different features and vectorize these features, and then utilize some rule-based and machine learning algorithms to detect anomalies. And the universal symbols are listed in Table 1.

4.1 Straggler Tasks

A straggler task is a task that executes longer time than other tasks in a stage. Because a stage cannot complete until all the tasks within it have completed, a straggler task significantly delays the completion time of the stage. For example, according to the report from [8], we know 25% stages have more than 15% outlier tasks. Note that the outlier task is the task that has more than 1.5 times runtime compared with the median task duration in its stage.

Therefore we define the detection rule of straggler tasks and straggler nodes as follows.

1) The task j of stage s_i is an straggler task, when $(D_j^{s_i}/median(D^{s_i})) > Th\_D$. Here, $median(D^{s_i})$ is the median task runtime duration in stage s_i. Th_D represents a predefined threshold of outlier tasks, which can be set as $Th\_D = 1.5$.

2) The node k is a straggler node, when $(\overline{D_k^{s_i}}/median(D^{s_i})) > Th\_D$. Here, $\overline{D_k^{s_i}}$ is the average task runtime of stage s_i on node k.

Table 1. Universal Symbols

Symbol	Description
J	Jobs of Big Data application
s_i	The i-th stage in job J
j	The j-th task in job J
$SNum_J$	Stage number of job J
k	Node k in cluster, $1 \leqslant k \leqslant p$, and p is the number of slaves in the cluster
$TNum_k^{s_i}$	Task number of stage s_i on node k
$dataSize_j^{s_i}$	In stage s_i, the data size that task j processes
$locality^l$	The l-th category of data locality
$D_j^{s_i}$	Runtime of task j in the stage s_i
$metric_{kn^*}$	The n^*-th performance metric on node k
Th_D	Threshold of straggler task
Th_UB	Threshold of jobs with task assignment imbalance
Th_size	Threshold of data size
Th_simi	Threshold of similarity

4.2 Task Assignment Imbalance

We consider the uneven task assignment as task assignment imbalance. For example, the tasks assigned to one node are much more or less than those to the other nodes in a stage. And we propose an algorithm to detect whether the task assignment of an application is balanced, which is described in Algorithm 1.

Algorithm 1. Determining Task Assignment Imbalance

Input:
$TNum_k^{s_i}$, BC, Th_UB, J
Output:
unbalanced s_i, unbalanced J
1: $Ratio\_UB = 0$;
2: $Count\_UB = 0$;
3: **for** $i = 1$; $i < n$; $i++$ **do**
4: Set $Diff^{s_i} = 0$;
5: **for** $k = 1$; $k < p$; $k++$ **do**
6: $Diff\_TNum_k = TNum_k^{s_i} - \overline{TNum^{s_i}}$;
7: $Diff^{s_i} = Diff^{s_i} + |Diff\_TNum_k|$;
8: **end for**
9: **if** $Diff^{s_i} > BC \times \overline{TNum^{s_i}} \times p$ **then**
10: Print 'task assignment at stage s_i is unbalanced'.
11: $Count\_UB++$;
12: **end if**
13: **end for**
14: $Ratio\_UB = \frac{Count\_UB}{SNum^{s_i}}$;
15: **if** $Ratio\_UB > Th\_UB$ **then**
16: Print 'task assignment of job J is unbalanced'.
17: **end if**

We first define $BC \times \overline{TNum^{s_i}}$ as the measurement of task assignment imbalance in stage s_i. The balance coefficient (BC) refers to the degree of workload imbalance which can be tolerated. $\overline{TNum^{s_i}}$ in (1) indicates the average number of tasks in stage s_i in the cluster (with p slaves) after removing ultrashort tasks, and $TNum_k^{s_i}$ refers to the volume of tasks in the stage s_i on the node k after removing ultrashort tasks. Since in some special cases, stages with a number of ultrashort tasks (e.g., failed tasks) running on nodes greatly affect the judgment of task assignment imbalance, we decide to eliminate the ultrashort tasks.

$$\overline{TNum^{s_i}} = \sum TNum_k^{s_i}/p. \tag{1}$$

In order to determine whether the task assignment imbalance exists in a stage, we define $Diff\_TNum_k$ to indicate the difference between $TNum_k^{s_i}$ and $\overline{TNum^{s_i}}$ on node k. If the absolute value $|Diff\_TNum_k|$ is greater than the number of tolerated imbalanced tasks $BC \times \overline{TNum^{s_i}}$, the workload on this node is considered as imbalanced. Moreover, if the sum $Diff^{s_i}$ is greater than $BC \times \overline{TNum^{s_i}} \times p$, we consider that the task assignment imbalance exists in a stage.

Further more, we assume that the stage number of the job J is $SNum^{s_i}$, and define $Ratio\_UB$ to indicate the ratio of imbalance stage in a job, which is the proportion of the unbalanced stage to the total stage number. If $Ratio\_UB > Th\_UB$ (here, Th_UB is a threshold of the job having task assignment imbalance, which can be set by users), we consider that the job J has workload imbalance.

4.3 Data Skew

Data skew mainly includes two situations: skew data size and uneven data placement.

4.3.1 Skew Data Size

The data size varies across tasks in a stage, and it will affect task runtime and lead to workload imbalance. For example, if the data is partitioned on a key space that has too little entropy, i.e., a few keys correspond to a lot of data, then the partitions will differ in sizes[8], and the corresponding tasks will run a long time because they have a lot of work to do.

From existing experience, the data skew phenomenon would exist in a task if the data size of the processing task is much larger or smaller than that of the other tasks in a stage. And a similar phenomenon exists in a node if the average size of data is far different from the other nodes in a stage. Furthermore, the data

skew of a stage would be deemed to be in existence if data skew was found in the task or node on this stage.

To measure the size of skew data, we choose the data size of each task processed $dataSize_j^{s_i}$ as the feature, and then we calculate the ratio of data size to the median value of data size $median(dataSize^{s_i})$. Afterwards the results of value comparison of the ratio and the threshold Th_size help us to measure the skew data size. And the detection rules of skew data size are as follows.

1) The data size of task j is skew, when $dataSize_j/median(dataSize^{s_i}) > Th\_size$, which can be used to determine whether an outlier task has data skew.

2) The node k has data size skew, when $\overline{dataSize_k}/median(dataSize^{s_i}) > Th\_size$, which can determine whether an outlier node has data skew.

4.3.2 Uneven Data Placement

The data placement is another critical factor for task imbalance. Because the hardware and workloads are different in the distributed cluster environment, the partitioned data may be placed unevenly, and the execution time of tasks can be very different. In order to find uneven data placement, we define data locality that refers to the data placement. And we focus on the impact of different data localities on the duration of the tasks.

First, we classify the runtime of tasks with some locality types into two categories: 1) normal execution time, and 2) abnormal execution time, which is defined as $oulierD_j^{s_i}$. Abnormal execution time represents much longer running time than the normal one. Second, we determine whether the data placement leads to abnormal execution time or not, by setting several different weights for distinct localities given the priority of locality. $locality^l$ and $pri(locality^l)$ are utilized here to infer to categories of locality and weights of locality respectively. For example, we set $pri(RACK\_LOCAL/NODE\_LOCALITY)$ to 1, $pri(ANY/OFF\_SWITCH)$ to 2, and some weights to 0 (0 means these localities are not supposed to cause uneven problems of the data placement).

Meanwhile, we calculate $Num(oulierD)_k^{locality^l}$, which refers to the number of abnormal runtime of $locality^l$ on node k. In addition, we define $Ratio(locality^l, k)$ in (2), and it indicates the ratio of uneven data placement on node k. When $Ratio(locality^l, k)$ is larger than 0, the uneven data placement occurs. The uneven data placement detection algorithm is based on Euclidean distance, and the whole procedure is demonstrated in Algorithm 2.

$$Ratio(locality^l, k) = \frac{Num(oulierD)_k^{locality^l}}{TNum_k^{s_i}} \times pri(locality^l). \quad (2)$$

Algorithm 2. Uneven Data Placement Detection Based on Euclidean Distance Outlier Algorithm

Input:
D^{s_i}, $locality^l$
Output:
$Ratio(locality^l, k)$
1: Calculate $median(D^{s_i})$, standard deviation $std(D^{s_i})$;
2: Calculate the distance dis from each $D_j^{s_i}$ to $median(D^{s_i})$: $dis_j = D_j^{s_i} - median(D_i^s)$;
3: **for** $j = 1; j < Num^{s_i}; j++$ **do**
4: Summarize: $sum(dis) = \sum(dis_j)$;
5: Calculate the mean value of dis: $mean(dis)$;
6: **if** $|dis_j| > mean(dis)$ **then**
7: Put $D_j^{s_i}$ into the suspicion group $SuspG$;
8: **end if**
9: **end for**
10: **for** each $D_j^{s_i}$ in $SuspG$ **do**
11: **if** $||dis_j| - mean(dis)| > std(D^{s_i}) \times 1.96$ **then**
12: Put $D_j^{s_i}$ into $outlier\_list(D)$;
13: Find the corresponding $node_j$ and $locality_j$ of $D_j^{s_i}$
14: **end if**
15: **end for**
16: $Ratio\_UP$=0;
17: **for** $k = 1; k < p; k++$ **do**
18: **for** each $locality^l$ in locality categories **do**
19: Calculate $Num(oulierD)_k^{locality^l}$;
20: Calculate $Ratio(locality^l, k)$;
21: **if** $Ratio(locality^l, k) > 0$ **then**
22: Output $Ratio(locality^l, k)$, and its corresponding node k and $locality^l$, which has uneven data placement
23: **end if**
24: **end for**
25: **end for**

4.4 Abnormal Node

Execution behaviors of various tasks at the same stage show a striking similarity while these tasks running on a homogeneous cluster; thus characteristics of nodes in the homogeneous cluster at one stage are supposed to be analogous. When a node shows significantly different characteristics compared with the other nodes at the same stage, the node would be regarded as an

abnormal node with potential bottlenecks. For example, machine failure or resource contention affects the task runtime and leads to workload imbalance. Also, if the disk loads on one node are much heavier than that of the other nodes, the reduce tasks on the node that has heavy disk loads are likely to be outliers.

In this subsection, we figure out cosine similarity between nodes to check abnormal machines. First, we convert collected performance metrics into metric vector $\boldsymbol{v}$. Here, $\boldsymbol{v}_k = \{avg(metric_{k1}), ..., avg(metric_{kn^*})\}$, $avg(metric_{kn^*})$ refers to the average value of $metric_{n^*}$, and n^* refers to the n^*-th collected metric. Then, we calculate the cosine similarity in the metric vector on between node k ($\boldsymbol{v}_k$) and node k^* ($\boldsymbol{v}_{k^*}$), as seen in (3). The closer the cosine value is to 1, the smaller the angle between two vectors and the more similar nodes we get.

$$simi(\boldsymbol{v}_k, \boldsymbol{v}_{k^*}) = \cos(\theta) = \frac{\boldsymbol{v}_k \cdot \boldsymbol{v}_{k^*}}{||\boldsymbol{v}_k|| \times ||\boldsymbol{v}_{k^*}||}. \quad (3)$$

For the sake of detecting abnormal nodes, we abandon the pairwise comparison method that lacks intuitive results. Instead, we measure the average similarity $simi(\boldsymbol{v}_k, \boldsymbol{v}_{\text{others}})$ between each node and all rest nodes shown in (4). If $simi(\boldsymbol{v}_k, \boldsymbol{v}_{\text{others}})$ of node k is smaller than a specified similarity threshold Th_simi, then the node k is regarded as an abnormal node. Here, $\{Slaves \setminus k\}$ refers to the slave nodes without node k.

$$simi(\boldsymbol{v}_k, \boldsymbol{v}_{\text{others}}) = \frac{\sum_{node \in \{Slaves \setminus k\}} simi(\boldsymbol{v}_k, \boldsymbol{v}_{\text{node}})}{p-1}. \quad (4)$$

4.5 Outlier Metrics

Generally, if there are abnormal nodes during a stage, by observing from metric level, individual metrics of abnormal nodes always have abnormal states. Even if some nodes are only subject to interferences, the interfered metrics will also behave differently. Therefore the metrics on one node have different behaviors from the metrics' behaviors on the other node, which can be regarded as outlier metrics. In this subsection, we compare the differences between the principal component metrics at each node in the cluster, and try to find the root cause of performance bottlenecks by observing the anomalies of metrics.

Here, we define performance metric matrix $\boldsymbol{X}$, which is an $m \times n^*$ matrix at a stage, column n^* refers to the number of collected metrics and row m is collection times during a stage, that is, each row in the matrix determines feature values in a particular timestamp during a stage, for example, $metric_{n^*}^{tm}$ refers to $metric_{n^*}$ at the timestamp tm.

$$\boldsymbol{X} = \begin{pmatrix} metric_{k1}^{t1} & metric_{k2}^{t1} & ... & metric_{kn^*}^{t1} \\ metric_{k1}^{t2} & metric_{k2}^{t2} & ... & metric_{kn^*}^{t2} \\ \vdots & \vdots & \vdots & \vdots \\ metric_{k1}^{tm} & metric_{k2}^{tm} & ... & metric_{kn^*}^{tm} \end{pmatrix}.$$

Based on the performance metric matrix $\boldsymbol{X}$, through principal component analysis, time series transformation, normalization and outlier detection, we could find the abnormal metrics.

4.5.1 Principal Component Analysis

According to the observations, we learn that not all performance metrics are closely associated with performance anomalies, for some metrics remain stable even if an outlier appears. Furthermore, different applications and stages are sensitive to different metrics; hence we use principal component analysis (PCA) for relevant metrics selection.

In general terms, PCA uses an orthogonal transformation to convert to a large set of data observations. The number of principal components is usually less than or equal to that of original variables, and the first principal component accounts for the most variability in the data. In addition, the cumulative contribution rate of PCA is used in our evaluation for selecting and determining an appropriate dimension of eigenspace. And in the experiment, we set the cumulative contribution rate to 95%.

4.5.2 Time Series Transformation

After principal component analysis, we perform time series transformation on the metric sequences. Here, we introduce two different methods of time-frequency transformation for input data, and the details of both two transformations are described as follows.

1) *Mean Value.* Average value comparison of the performance metric on different nodes is a typical approach for time series transformation. If there are substantial differences in average value of one performance metric between certain nodes and the other nodes, then we believe that this performance metric is a potential key metric, and the calculation method is shown in (5).

$$mean(metric_{kn^*}^{s_i}) = \frac{\sum_{tm=1}^{N} metric_{kn^*}^{tm}}{N}. \quad (5)$$

2) *Fast Fourier Transform.* Fast Fourier transform (FFT)[12] is often utilized to transform original data from time-space domain to frequency domain and vice versa, which is an efficient method of discrete Fourier transform (DFT). Moreover, FFT rapidly computes such transformations by factorizing the DFT matrix into a product of sparse (mostly zero) factors. As a result, FFT manages to reduce the complexity of computing DFT from $O(n^2)$, which arises if one simply applies the definition of DFT, to $O(n \log n)$, where n is the data size. After the fast Fourier transform of metric sequence, we will get $fft(metric_{kn^*}^{s_i})$.

4.5.3 Normalization

Different performance metrics in a cluster normally have varied sizes and units. For instance, the units of *cpu_usage* and *mem_usage* are percentage (%), and its value is between zero and one. However, the units of *diskR_band* and *netS_band* may be MB/s, which is different with the percentage. To adjust metrics measured at the stage on different scales to a common scale notionally, normalization is applied into the data preprocessing, and with the help of that, it would be more normal to process the data with consistent statistical properties.

In this subsection, we use the linear min-max normalization to convert the original metrics into values ranging from 0 to 1. (6) is the transformation expression, y is a sample in $mean(metric_{kn^*}^{s_i})$ or $fft(metric_{kn^*}^{s_i})$, max is the maximum value of the samples, and min is the minimum value of the samples. However, the disadvantage of this normalization method is that max and min might be redefined when inputting the extra new data.

$$y^* = \frac{y - min}{max - min}. \tag{6}$$

4.5.4 Outlier Detection

In statistics, an outlier is an observation point that is distant from other observations. In this subsection, we propose an unsupervised method combing with distance and magnitude for outlier detection to distinguish the metrics that do not belong to any expected pattern in the dataset or show certain similarities to other metrics.

Our distance-based outlier model borrows ideas from the distribution-based approaches. It is also suitable for situations where the observed distribution does not fit any standard distribution[13]. Specifically, an object o in a dataset D is an $DB(pct, dmin)$-outlier, if at least a fraction pct of all data objects in D lies at a distance greater than the threshold $dmin$, from o. We use the term $DB(pct, dmin)$-outlier as the shorthand notation for a distance-based outlier (detected using parameters pct and $dmin$). Of course, the choice of parameters pct and $dmin$, and validity checking (i.e., deciding whether each $DB(pct, dmin)$-outlier is a real outlier), require expert knowledge or experience.

Even though our distance-based outlier algorithm with appropriate parameter settings is able to detect most of outliers in the dataset, some outliers could still be missed. For instance, the normalized mean value of *cpu_usage* on each node is [hw073: 0.006 838, hw106: 0.156 043 99, hw114: 0.178 105 99]. However, there would exist no outlier as setting $dmin$ equal to 0.5 and pct equal to 1. Actually, we could consider 0.006 838 as an outlier value here. To make our outlier detection model still work in this case, we apply the logarithm method (e.g., log(10)) in the beginning to obtain data's order of magnitude by transforming the original data. For example, the order of magnitude on several nodes [hw073: -2, hw106: 0, hw114: 0] shows significant disparity and we can suppose hw073 is a potential outlier, then the remaining two nodes would be analyzed through the distance-based detection algorithm.

Algorithm 3 is the detailed pseudo-code, and describes the outlier detection algorithm based on distance and magnitude. In this algorithm, we predefine the parameter pct with a default value 1, and $dmin$ is adjustable. In addition, we use two methods to calculate the representative point of class A or B. One method is to compute the maximum/minimum value of the larger class, and the other method is to compute the median value of the larger class. In the subsequent experiments, we will compare the results of outlier detection by the maximum/minimum value method and the median value method with different $dmin$ for the larger class.

4.6 Summary

In addition, since the performance data and the anomalies at the application level are related, the system performance metrics are also helpful for detecting straggler tasks, task assignment imbalance, data size skew and uneven data placement. The application logs can be used to discover abnormal nodes, too, for example, extracting the corresponding features and building the anomaly detection models, which can be used to detect abnormal nodes. In the future work, we will

propose more anomaly detection methods to build a comprehensive library of anomaly detection.

Algorithm 3. Outlier Detection Algorithm Based on Distance and Magnitude

Input:
normalized $meanSet(metric^{s_i})$ or $fftSet(metric^{s_i})$
Output:
outlier metrics
if $min - max \leqslant -2\log(10)$ **then**
  log(10) conversion for input dataset
  Call the magnitude-based outlier algorithm:
  1) Find the center of mass (median);
  2) Calculate the distance dis from each point to the center of mass
  if $dis > avg(dis)$ **then**
    Add the point into the suspicion group $SuspG$.
  end if
  3) Compute the distance $dis(SuspG)$ from the point in $SuspG$ to the center of mass;
  if $(dis(SuspG) - avg(dis)) > variance$ **then**
    This point in $SuspG$ is counted as outlier;
  end if
else
  Call the distance-based outlier algorithm:
  1) Select the maximum and minimum values for the current point in classes A and B;
  2) Calculate the distance $dis(A)$ and $dis(B)$ from each point to the two current points;
  if $dis(A) < Th\_knn$ **then**
    Assign the point to class A;
  else
    Assign the point to class B;
  end if
  if $Num(A) < Num(B)$ **then**
    3) Compute the distance $dis(a, B)$ from the point a in A to the class B (the representative point of class B);
    if $dis(a, B) > dmin$ **then**
      This point a is counted as outlier.
    end if
  else
    3) Compute the distance $dis(b, A)$ from the point b in B to the class A (the representative point of class A);
    if $dis(b, A) \geqslant dmin$ **then**
      This point b is counted as outlier.
    end if
  end if
end if

5 HybridTune Implementation

Based on our general performance diagnosis approach, we have implemented HybridTune, a scalable, lightweight, model-based and data-driven performance diagnosis tool utilizing spatio-temporal correlation. In this section, we describe the implementation of HybridTune, and the workflow of HybridTune is shown in Fig.3.

5.1 Data Collection

We use the data collector of BDtune[11] to gather the performance information and application logs from the software stack of Big Data systems at different levels. Specifically, the data collector collects architecture-level metrics, system-level metrics and application logs. Hardware performance monitoring unit (PMU) and Perf① are used for data sampling of architecture-level metrics in the data collector, and metrics consist of instruction ratio, instructions per cycle (IPC), cache miss, translation lookaside buffer (TLB) miss, etc. Then, we use Hmonitor[11] to collect raw data from the filesystem/proc, which provides key parameters (e.g., CPU usage, memory access, disk I/O bandwidth, network bandwidth) of system performance. Furthermore, we use log collection tools to collect the application logs (e.g., history job logs of Spark and Hadoop).

5.2 Data Preprocessing

Data preprocessing is an important step, since log files and performance data with non-uniform formats are generally collected from different nodes. Therefore, we parse the performance data and application logs, and then unify the data format, preprocess the raw data and load the data into our MySQL database.

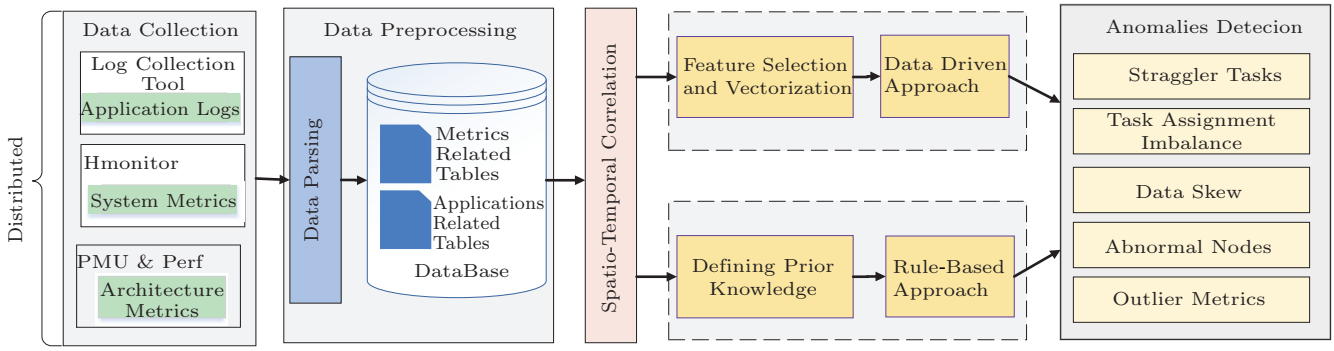


Fig.3. Workflow of HybridTune.

①https://perf.wiki.kernel.org/index.php/Main_Page, Sept. 2019.

5.2.1 Data Parsing

In order to deal with different log formats of applications, we establish various log parsing templates compatible with different applications' logs. In our implementation, we collect the history job logs of Hadoop and Spark, which are JSON formats and record various information about jobs' run. Then we parse and extract some useful application data from these history job logs, which include: 1) runtime information: the submission time, completion time and runtime of jobs, stages and tasks; 2) dataflow information: the data flow information between nodes in each stage of jobs, including reading and writing data, reading and writing time, input and output data of tasks, and so on; 3) application configuration information: the configuration parameter information of Hadoop/Spark, etc; 4) job runtime parameters: job-level parameters and task-level parameters, for example, the "counters" information of Hadoop and the "task_metrics" information of Spark.

Simultaneously, we use the collected raw metrics to calculate the selected performance metrics which are shown in Table 2. For different performance metrics, there are different calculation methods, for instance, *cpu_usage* can be calculated by metrics of *usr*, *nice*, *sys*, *idle*, *iowait*, *irq* and $softirq$.

5.2.2 Data Storage

In addition, we design a tagging mechanism and propose an incremental table approach to match the scalability need of date aggregation and storage.

Specifically, we set corresponding labels for tables of different applications. For instance, if $Type\_Flag = 0$, then the tables represent the parsed Hadoop log contents; if $Type\_Flag = 1$, we know that tables store the parsed Spark logs. Moreover, we provide public table interfaces and unique table interfaces for different application logs, because the contents of application logs are not always the same. For example, both Hadoop and Spark consist of public table interfaces like *app* table, *job* table, *stage* table and *task* table. The unique table interfaces for Hadoop are *task_attempt* table and *counters* table. Spark's unique table interfaces are *rdd* table and *task_metrics* table. When collecting Storm logs and storing parsed data into MySQL, the data preprocessor needs to adjust its log parsing template only for Storm applications, and then it creates unique tables of Storm, parses data, preprocesses raw data, loads data into existed public tables and its unique tables, and sets $Type\_Flag$.

Table 2. Selected Typical Performance Metrics

Layer	Metrics	Description
System level	cpu_usage	CPU utilizations
	mem_usage	Memory usage
	ioWaitRatio	Percentage of CPU time spent by IO wait
	weighted_io	Average weighted disk io time
	diskR_band	Disk read bandwidth
	diskW_band	Disk write bandwidth
	netS_band	Network send bandwidth
	netR_band	Network receive bandwidth
Architecture level	IPC	Instructions per cycle
	L2_MPKI	Misses per kilo instructions of L2 cache
	L3_MPKI	Misses per kilo instructions of L3 cache
	L1I_MPKI	Misses per kilo instructions of L1I cache
	ITLB_MPKI	Misses per kilo instructions of ITLB
	DTLB_MPKI	Misses per kilo instructions of DTLB
	MUL_Ratio	MUL operations' percentage
	DIV_Ratio	DIV operations' percentage
	FP_Ratio	Floating point operations' percentage
	LOAD_Ratio	Load operations' percentage
	STORE_Ratio	Store operations' percentage
	BR_Ratio	Branch operations' percentage

5.3 Anomalies Detection

The performance anomalies detection module of HybridTune is equipped with a plug-in mechanism, which enables the analysis engine to adapt different application occasions and different diagnosis methods. Among these plug-in mechanisms, some are universal (e.g., statistical analysis of performance metrics), while some are application-specific (e.g., critical path computing of different jobs).

Specifically, we build the corresponding rules or algorithms to detect the anomalies. For instance, the detection methods of straggler tasks, task assignment imbalance and skew data size are based on prior rules. The detection methods of uneven data placement, abnormal nodes, outlier metrics are based on the improved machine learning algorithms. For example, uneven data placement detection is based on the Euclidean distance outlier algorithm, abnormal node detection is based on cosine similarity, and outlier detection is based on distance and magnitude. And the details have been described in Section 4.

6 Evaluations

6.1 Experiment Settings

The Hadoop cluster used in our experiment consists of one master machine and six slave machines, and the Spark cluster is deployed on the Hadoop Yarn framework. In our cluster, we use the NTP (Network Time Protocol) service to ensure clock synchronization across nodes, and each compute node has a hardware configuration in Table 3. In addition, the evaluations about impacts of configurations on system performance are not included in this paper; thus our machines in the cluster are homogeneous machines with the same machine configurations and cluster configuration parameters.

Table 3. Server Configurations

Component	Description
Processor	Intel® Xeon® CPU E5645@2.40 GHz
Disk	8 Seagate Constellation ES (SATA 6 Gb/s)-ST1000NM0011 [1.00 TB]
Memory	32 GB per server
Network	Broadcom Corporation NetXtreme II BCM5709 Gigabit Ethernet (rev 20)
Kernel	Linux Kernel 3.11.10

6.2 Anomaly Simulation

To further determine whether the workload imbalance, straggler nodes, data skew and abnormal machine states exist, and evaluate the effectiveness of our automatic diagnosis tool for performance bottleneck detection, we decide to simulate anomalies by the following methods.

1) *Reducing the Computing Power of Some Nodes.* For example, you can attempt to disable a core in a multi-core machine. However, this does not work for a few certain cores, like the core 0. And disabling too many cores would result in a system crash.

2) *Making Disk Storage Imbalance.* In our experiments, we fill the space of disks on some nodes with data.

3) *Mixing Interference Workloads.* In our experiments, Linux stress testing tool Stress② is utilized to impose extra load on CPU, memory, IO and disk.

4) *Cache Flusher Adjustment.* We use a cache flusher to load the certain volume of data according to the size of the last-level cache, for inserting cache anomalies.

6.3 Runtime Overheads

Because data collector needs to gather multi-level metrics through PMU, Perf and HMonitor on each node, while the aggregation and the analysis of collected data are performed on a separated node in an offline fashion, the data collector is the major source of runtime overheads in HybirdTune.

We first measure the baseline completion time when jobs are running without the data collector. Then we set the interval of collecting the system-level metrics and architecture-level metrics as one second, and measure the instrumented completion time when jobs are running with the data collector. Fig.4 shows the ratio of the instrumented completion time over the baseline completion time for selected benchmarks in BigdataBench suite[7] (WordCount, Sort, Grep of Hadoop and Spark). The datasets of WordCount and Grep are 60 GB and 120 GB respectively, which are from the text data generater of BDGS[14], and the dataset of Sort is 60 GB, which is from the sort data generater of BDGS. It is clear that the additional overhead caused by HybirdTune is very low for the evaluated benchmarks. It can be seen that the data collection tool of HybirdTune is a lightweight, low-overhead, non-intrusive tool, which does not bring great impact on Big Data systems.

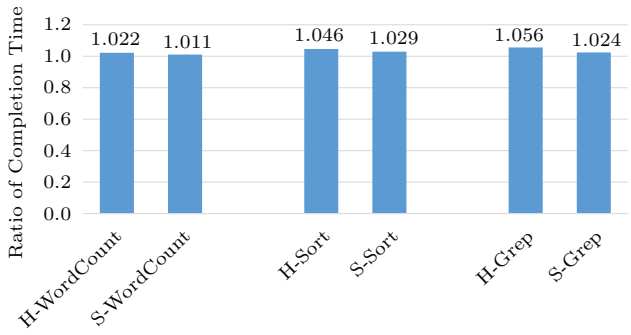


Fig.4. Ratio of the instrumented completion time over the baseline completion time. H: Hadoop; S: Spark.

6.4 Anomalies Detection Evaluation Results

In this subsection, over 90 programs on 342 stages are tested and executed. We also try a number of workload executions, e.g., Wordcount, Grep, Sort, *K*-means on BigDataBench, FPGrowgh and PrefixSpan on Spark MLlib[15]. Due to that the bottlenecks at the application level (such as workload imbalance and data skew) may be related to the users' subjective view

②http://people.seas.harvard.edu/∼apw/stress/stress-1.0.4.tar.gz, Sept. 2019.

even more, we plan to use the artificially setting thresholds. For example, we can set $Th\_D = 1.5$, $BC = 0.1$, $Th\_UB = 0.6$ and $Th\_size = 1.5$[11]. Here, we mainly evaluate the threshold selection of abnormal nodes and outlier metrics, as well as the effect of outlier detection.

Due to the rule-based detection methods and detection algorithm of uneven data placement, we have set the corresponding thresholds according to experience. Therefore in this subsection, we mainly evaluate the effectiveness of abnormal node and outlier metrics detection with different thresholds.

6.4.1 Determining Abnormal Node

To determine whether or not node k is an abnormal node, we compare $simi(\boldsymbol{v}_k, \boldsymbol{v}_{\text{others}})$ ((4) in Subsection 4.4) with the predefined threshold Th_simi. If the similarity value is smaller than the threshold, then the node is judged as an abnormal node. As for how to set the size of threshold Th_simi, we utilize (7) to measure the proportion of real abnormal nodes in the detected abnormal nodes detected just by predefined threshold Th_simi, and the results are shown in Table 4.

$$Ratio(Ab\_Node) = \frac{\text{number of abnormal nodes}}{\text{number of detected abnormal nodes}}. \quad (7)$$

6.4.2 Effect of Outlier Detection

We evaluate the effectiveness of outlier metrics detection through three indicators: *Precision*, *Recall* and *F*1-*Score* ((8), (9) and (10) respectively)[16].

$$Precision = \frac{\text{number of successful detections}}{\text{number of total alarms}}, \quad (8)$$

$$Recall = \frac{\text{number of successful detections}}{\text{number of total outliers}}, \quad (9)$$

$$F1\text{-}Score = \frac{2 \times Precision \times Recall}{Precision + Recall}. \quad (10)$$

Table 4. Proportion of Real Abnormal Nodes in the Detected Abnormal Nodes

Th_simi	$Ratio(Ab\_Node)$ (%)
0.40	96.5
0.45	92.5
0.50	83.8
0.55	69.4
0.55	65.7
0.55	54.1
0.55	42.1

We see the similar results of the outlier metrics detection by mean-value transformation and FFT transformation from Fig.5 and Fig.6 respectively. If we utilize the median value to represent the larger class, then *Precision* is higher than *Recall*. As shown in Fig.5 and Fig.6, when *dmin* equals 0.5 or 0.6, *Precision* reaches more than 92%, however *Recall* is slightly lower: 67% and 70% respectively. In addition, if using maximum/minimum value as the larger class, in contrast, *Recall* would be higher than *Precision*, then we see *Recall*s in Fig.5 and Fig.6 are both over 84% no

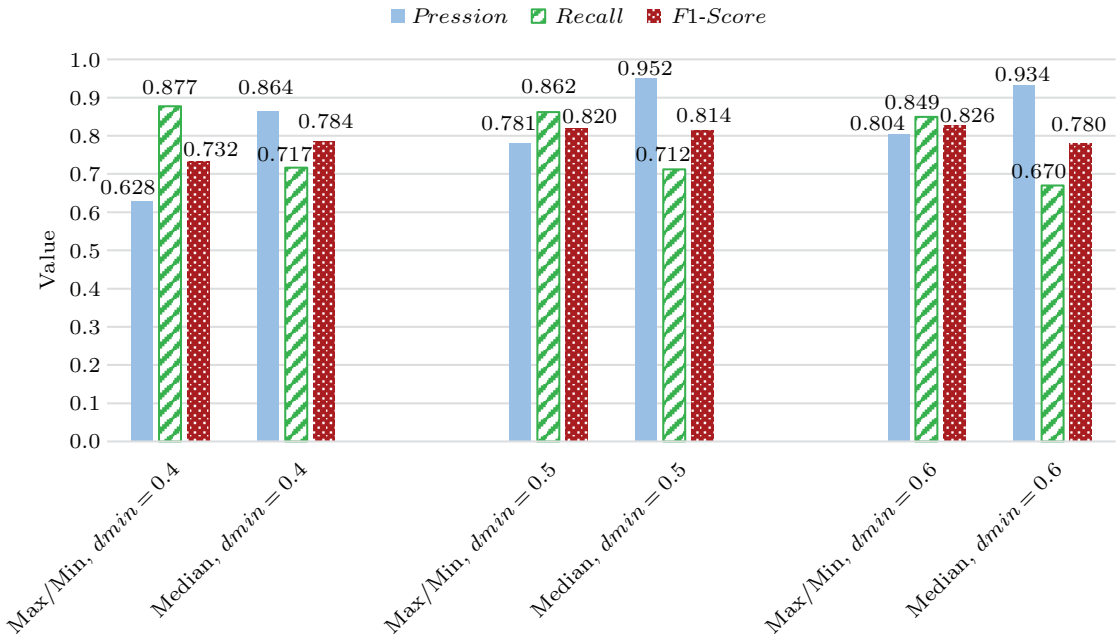


Fig.5. Effectiveness of outlier detection by using mean-value transformation.

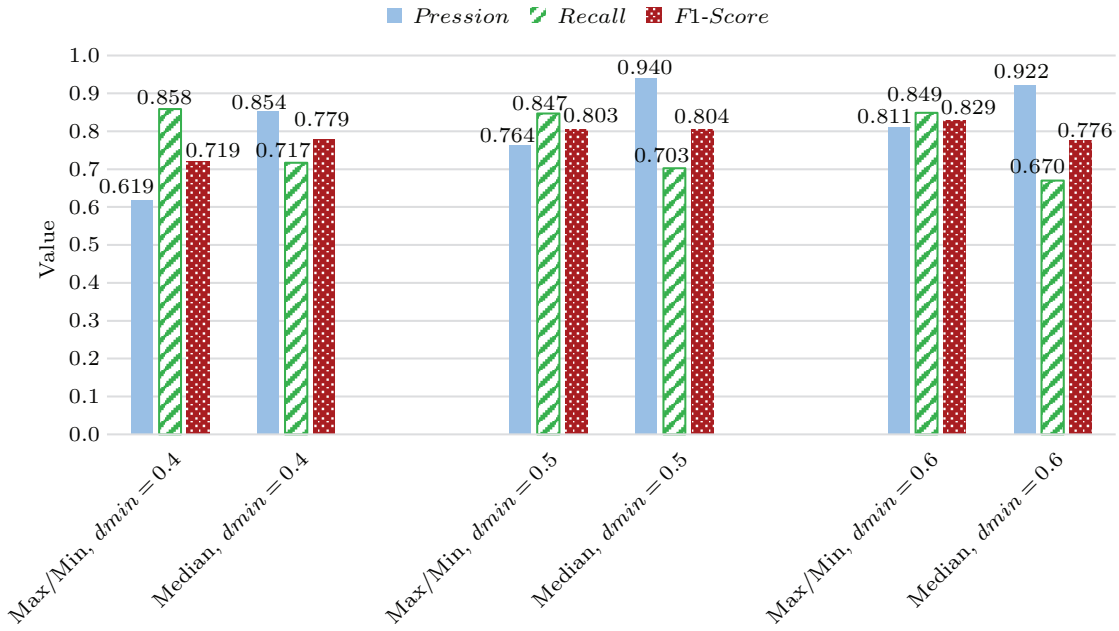


Fig.6. Effectiveness of outlier detection by using FFT transformation.

matter what *dmin* is. In other words, there are more outliers able to be detected and the false negative rate is much lower. Contrary to *Recall*, *Precision* is a little low. If the *dmin* value is 0.4, 0.5 or 0.6, then *Precision* ranges from 62% to 81%, which means that lots of normal metrics are misjudged as outliers, thus resulting in a high false positive rate. Additionally, when the maximum/minimum value represents the larger class and *dmin* is 0.6, then *Accuracy* closes to 83%. However, if *dmin* is set to 0.5, *Accuracy* is about 80% when using the maximum/minimum value and the median value, and setting *dmin* to 0.4 would further decrease *Accuracy*.

7 Case Studies

In this section, based on our detection and diagnostic methods, we share our experiences on tuning and diagnosing the performance of Spark and Hadoop applications. In detail, we illustrate the three Spark cases which have reported in [11] and a case for Hadoop job.

7.1 Case-1: Uneven Data Placement

From [11] we know that the S-WordCount job's stage *spark_stage_app-20160630230531-0000_0* has a straggle outlier node hw114 when $Th\_D = 1.5$, and workload imbalance when $BC = 0.1$. In this paper, we give the automatic diagnostic results in Fig.7.

In contrast to the other nodes, the priority of data locality of the hw114 node is "ANY" and the average similarity between hw114 and the other slave nodes is 23.57%, because the uneven data phenomenon placement exists in hw114. We find that every task on node hw114 needs to read data from the other nodes rather than the local, so that their task runtime is relatively longer. Therefore we decide to utilize the HDFS (Hadoop Distributed File System) balancer to optimize the data distribution, and then the completion time of this S-WordCount job is reduced from 218 seconds to 167 seconds, approximately 23.21%.

Stage id: *spark_stage_app-20160630230531-0000_0*
Detected straggle outlier node: hw114
Detected workload imbalance: hw114
— ***Data skew diagnosis.***
Skew data size: Null
Uneven data placement : hw114 [ANY:0.06875]
— ***Abnormal node diagnosis.***
Similarity analysis: Similarity ([hw089, hw062, hw073, hw103, hw114, hw106], other nodes): [0.8048, 0.7838, 0.8242, 0.7870, 0.2359, 0.8171]
Detected abnormal node: hw114
— ***Outlier metrics diagnosis***:
Mode: [Mean-Value,median, $CCRate_d = 0.95$, dmin = 0.5]: hw114:(mem_usage, ioWaitRatio, diskR_band, netS_band, netR_band)

Fig.7. Automatic diagnostic results of Spark job for case-1.

In addition, we also give an automatic diagnostic results of Hadoop's *mapStage_job_1493084522519_0014* in Fig.8. We find that this map stage of Hadoop has no

obviously straggle outlier node, but there exist workload imbalances. In fact, we check the task assignments of hw106, hw114, hw062 and hw073, which are 228, 159, 44 and 23 respectively. It is obvious that the assigned tasks in hw073 and hw062 are significantly less than that in the other two nodes, and the reason is that the localities of hw073 and hw062 are "RACK_LOCAL", while the localities of hw106 and hw114 are "NODE_LOCAL".

Stageid: *mapStage_job_1493084522519_0014*
Detected straggle outlier node: Null
Detected workload imbalance: hw106, hw073, hw062, hw114
— ***Data skew diagnosis.***
Skew data size: Null
Uneven data placement : hw073[RACK LOCAL:0.09469], hw062 [RACK LOCAL:0.00379]
— ***Abnormal node diagnosis***:
Similarity analysis: Similarity ([hw062, hw073, hw114, hw106], other nodes): [0.8571, 0.8143,0.8784, 0.8807]
Detected abnormal node: Null
— ***Outlier metrics diagnosis***:
Mode: [Mean_Value, median, $CCRate_d = 0.95$, dmin = 0.5]: Null

Fig.8. Automatic diagnostic results of Hadoop job for case-1.

7.2 Case-2: Abnormal Node

We also give the automatic diagnostic results of the case *spark_stage_app-20160719212517-0001_2* (reported in [11]) in Fig.9. From the automatic diagnostic results we know that hw089 is a straggle outlier node and has workload imbalance, and it is a abnormal node whose similarity between the others is about 11.98%. Nevertheless, these bottlenecks are not mainly caused by data skew. However, we do find some outlier metrics, and just find an abnormal metric: the average *weighted_io* of hw089 calculated by mean-value method is −4 177 890.23, contrary to common sense. The *io_time_weighted* value is 4 294 936 240 at 22:35:49 2016-07-19, and the *io_time_weighted* value is 258 900 at 22:35:50 2016-07-19.

In order to diagnose the root cause, we further view the system logs, and find that the disk of hw089 has experienced a high temperate alarm and raw read error rate[11]. Therefore we think that it is necessary to analyze the correlation between the outlier metrics, for some outlier metrics may be caused by other metrics, such as case-2, where the abnormal metric *weighted_io* leads to the outlier metrics *cpu_usage* and *ioWaitRatio*. Furthermore, in order to locate the root cause of abnormal metrics, the diagnosis based on the system or RAS logs is also needed.

Stage id: *spark_stage_app-20160719212517-0001_2*
Detected straggle outlier node: hw089
Detected workload imbalance: hw089
—***Data skew diagnosis***:
Skew data size: Null
Uneven data placement: Null
—***Abnormal node diagnosis***:
Similarity analysis: Similarity ([hw089, hw062, hw073, hw103, hw114, hw106], other nodes): [0.1198, 0.7667,0.8017, 0.7774, 0.7995, 0.7974]
Detected abnormal node: hw089
—***Outlier metrics diagnosis***:
Mode.[Mean-Value, median, $CCRate_d = 0.95$, dmin = 0.5]: hw089: (cpu_usage, ioWaitRatio, weighted io)

Fig.9. Automatic diagnostic results of Spark job for case-2.

7.3 Case-3: Intra-Node Resource Interference

The automatic diagnostic results of the case *spark_stage_app-20160703145107-0001_0* (reported in [11]) are in Fig.10. From the automatic diagnostic results we know that, there exist two straggle outlier nodes hw062 and hw106, and they are caused neither by data skew nor abnormal nodes. However, the automatic diagnosis tool of BDTune finds that the average values of $L3\_MPKI$ in these two nodes are both larger than those in the other nodes while the node similarity of all nodes in the cluster is 93.3%.

Stageid: *spark_stage_app-20160703145107-0001_0*
Detectedstraggleoutliernode: hw062, hw106
Detectedworkloadimbalance.
—***DataSkewdiagnosis.***
Skewdatasize: Null
Unevendataplacement : Null
—***Abnormalnodediagnosis.***
Similarityanalysis: Similarity ([hw089, hw062, hw073, hw103, hw114, hw106], other nodes): [0.9593, 0.9255,0.9228, 0.9437, 0.9513, 0.9432]
Detectedabnormalnode: Null
—***Outliermetricsdiagnosis.***
Mode: [FFT, median, $CCRate_d = 0.95$, dmin = 0.5]: hw062:(L3_MPKI); hw106:(L3_MPKI)

Fig.10. Automatic diagnostic results of Spark job for case-3.

8 Related Work

Performance Analysis. There have been many prior studies on building tools to analyze performance for MapReduce applications. SONATA[2] provides a correlation-based performance analysis approach for full-system MapReduce optimization. It correlates

different phases, tasks and resources for identifying critical outliers and recommends optimization suggestions based on embedded rules, which just uses the model-based method. HiTune[1], a dataflow-driven performance analysis approach, reconstructs the high-level, dataflow-based, distributed and dynamic execution process for each Big Data application. Mochi[4] is a visual, log-analysis based debugging tool that correlates Hadoop's behavior in space, time and volume, and extracts a causal, unified control and dataflow model of Hadoop across the nodes of a cluster.

Besides the above tools used to analyze MapReduce applications, tools for other platforms are also proposed. Ousterhout *et al.*[17] used blocked time analysis to quantify the performance bottlenecks in Spark framework, and Microsoft used Artemis[5] to analyze Dryad application, which is a plug-in mechanism which uses statistical and machine learning algorithms. Roots[18] is a full-stack monitoring and analysis system for performance anomaly detection and bottleneck identification in cloud platform-as-a-service (PaaS) systems. Table 5 shows the comparison of performance analysis tools for Big Data systems.

Performance Anomaly Detection and Diagnosis. In general, anomaly detection is an essential part of performance diagnosis for Big Data systems. And anomaly detection techniques can be classified into two methods: data-driven and model-based. Data-driven methods include nearest neighbor based methods including distance-based[13], k-nearest neighbor[19], local outlier factor[20], and k-means clustering[21]. Specifically, a number of node comparison methods have been adopted for anomaly detection in large-scale systems[22]. For example, Kahuna[23] aims to diagnose performance in MapReduce systems based on the hypothesis that nodes exhibit peer-similarity under fault-free conditions and some faults result in peer-dissimilarity. Ganesha[24] is a black-box diagnosis technique that examines OS-level metrics to detect and diagnose faults in MapReduce systems, and especially can diagnose faults that manifest asymmetrically at nodes. Eagle[25] is a framework for anomaly detection at eBay, which uses density estimation and PCA algorithms for user behavior analysis. Kasick *et al.*[26] developed anomaly detection mechanisms in distributed environments by comparing system metrics among nodes. Yu and Lan[22] presented a practical and scalable anomaly detection method for large-scale systems, based on hierarchical grouping, non-parametric clustering, and two-phase majority voting.

Table 5. Comparison of Performance Analysis Tools for Big Data Systems

Tool Name	Target System	Collection Tool	Data Type	Analytical Method	Extensibility	Timeliness
HybirdTune	Hadoop, Spark	PMU, Perf, HMonitor	Architecture metrics, system metrics, application logs	Spatio-temporal data and model driven diagnosis	Yes	Semi real-time
SONAT	MapReduce	Ganglia	Resource metrics, application traces	Correlation analysis, rules-based optimization	No	Offline
HiTune	Hadoop	Chukwa	Hadoop logs	Dataflow-driven analysis	No	Offline
Theia	Hadoop	Ganglia	Logs, performance data	Visual analysis	No	Offline
Mochi	Hadoop	SALSA	Logs	Control- and dataflow model analysis	No	Offline
Artemis	Dryad	Data-collection front-end	Dryad logs, performance counters data, XML data, binary data	Diagnosis plug-ins based on statistical and machine learning algorithms	No	Offline
Trace-analysis	Spark	Instrumentation	Spark traces	Blocked time analysis	No	Offline
Ganesha	MapReduce	Sysstat's sadc program	OS-level metrics	Black box diagnosis	No	Offline
Hadoop vaidya	MapReduce	Logging tool	MapReduce job data	Rule-based diagnosis	No	Offline
Eagle	Hadoop	Logging tool	Audit log	Density estimation, PCA	No	Real-time
Mantri	MapReduce (Dryad)	Scope compiler, cosmos scheduler	MapReduce job logs	Outlier analysis and optimization	No	Real-time

Representative model-based techniques include rule based methods[27], support vector machine (SVM) based methods[28], probability model[29], Bayesian network based methods[30], etc. For example, Hadoop vaidya③ is a rule-based performance diagnostic tool from MapReduce jobs. Although it can provide recommendations based on the analysis of runtime statistics, it cannot facilitate full-system optimization. CloudDiag[31] can efficiently pinpoint fine-grained causes of the performance problems through a black-box tracing mechanisms and without any domain-specific knowledge. Mantri[8] is a system that monitors tasks and culls outliers based on their causes, and then delivers the effective mitigation of outliers in MapReduce networks. Jia *et al.*[32] presented an approach of diagnosing anomalous run-time behaviors in distributed services from execution logs, and they mined a hybrid graph model including a service topology and a time-weighted control flow graph (TCFG) that captures healthy execution flows of each service. Ren *et al.*[33] proposed an online anomaly detection detection approach based on stage-task behaviors modeling.

Moreover, the pure data driven diagnosis approach is promising for relatively simple distributed applications, but it is very time-consuming and difficult to be used in the complex Big Data systems. The model-driven approach requires more detailed prior knowledge to achieve better accuracy, and it is also difficult to adapt for Big Data scale. Distinguished from the above work, HybridTune is a lightweight and extensible tool, which uses a hybrid method via combining the data-driven and the rule-based analysis approach. It provides fine-grained spatio-temporal correlation analysis and different diagnosis. Due to the stage-based and multi-level performance data correlation, it can be easily extended to semi-realtime detection and can improve the time effectiveness of diagnosis.

9 Conclusions

In this paper, taking advantage of the stage similarity and the distributed characteristics of Big Data systems, we analyzed the behaviors of the Big Data applications in terms of both system and architectural metrics of each stage, and proposed a hybrid approach that utilizes prior knowledge rules or data-driven machine learning algorithms to detect performance anomalies, such as straggler tasks, task assignment imbalance, data skew, abnormal nodes and performance outlier. In addition, we designed and implemented a lightweight, extensible tool HybridTune, and then validated its overhead and anomalies detection effectiveness. The overhead caused by HybridTune is just 5%, the accuracy of outlier detection could reach 93%. Then we reported several use cases, which show that our approach can pinpoint performance bottlenecks and provide performance optimization recommendations for Big Data applications efficiently.

Acknowledgement We are very grateful to anonymous reviewers.

References

[1] Dai J, Huang J, Huang S, Huang B, Liu Y. HiTune: Dataflow-based performance analysis for big data cloud. In *Proc. the 2011 USENIX Conference on USENIX Annual Technical Conference*, June 2011, Article No. 27.

[2] Guo Q, Li Y, Liu T, Wang K, Chen G, Bao X, Tang W. Correlation-based performance analysis for full-system MapReduce optimization. In *Proc. the 2013 IEEE International Conference on Big Data*, October 2013, pp.753-761.

[3] Garduño E, Kavulya S P, Tan J, Gandhi R, Narasimhan P. Theia: Visual signatures for problem diagnosis in large Hadoop clusters. In *Proc. the 26th Large Installation System Administration Conference*, December 2012, pp.33-42.

[4] Tan J, Pan X, Kavulya S, Gandhi R, Narasimhan P. Mochi: Visual log-analysis based tools for debugging Hadoop. In *Proc. USENIX Workshop on Hot Topics in Cloud Computing*, June 2009, Article No. 1.

[5] Cretu-Ciocarlie G, Budiu M, Goldszmidt M. Hunting for problems with Artemis. In *Proc. the 1st USENIX Workshop on Analysis of System Logs*, Dec. 2008, Article No. 2.

[6] Herodotou H, Lim H, Luo G, Borisov N, Dong L, Cetin F, Babu S. Starfish: A self-tuning system for big data analytics. In *Proc. the 5th Biennial Conference on Innovative Data Systems Research*, January 2011, pp.261-272.

[7] Wang L, Zhan J, Luo C, Zhu Y, Yang Q, He Y, Gao W, Jia Z, Shi Y, Zhang S, Zheng C, Lu G, Zhan K, Qiu B. BigDataBench: A Big Data benchmark suite from internet services. In *Proc. the 20th IEEE International Symposium on High Performance Computer Architecture*, February 2014, pp.488-499.

[8] Ananthanarayanan G, Kandula S, Greenberg A, Stoica I, Lu Y, Saha B, Harris E. Reining in the outliers in Map-Reduce clusters using Mantri. In *Proc. the 9th USENIX Conference on Operating Systems Design and Implementation*, October 2010, pp.265-278.

[9] Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, Franklin M, Shenker S, Stoica I. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proc. the 9th USENIX Symposium on Networked Systems Design and Implementation*, April 2012, pp.15-28.

③http://hadoop.apache.org/docs/r1.2.1/vaidya.html, Sept. 2019.

[10] Isard M, Budiu M, Yu Y, Birrell A, Fetterly D. Dryad: Distributed data-parallel programs from sequential building blocks. In *Proc. the 2007 EuroSys Conference*, March 2007, pp.59-72.

[11] Ren R, Jia Z, Wang L, Zhan J, Yi T. BDTUne: Hierarchical correlation-based performance analysis and rule-based diagnosis for big data systems. In *Proc. the IEEE International Conference on Big Data*, Dec. 2016, pp.555-562.

[12] Cochran W, Cooley J, Favin D, Helms H, Kaenel R, Langa W, Maling G, Nelson D, Rader C, Welch P. What is the fast Fourier transform? *IEEE Transactions on Audio and Electroacoustics*, 1967, 55(10): 1664-1674.

[13] Knorr E M, Ng R T. Algorithms for mining distance-based outliers in large datasets. In *Proc. the 24th International Conference on Very Large Data Bases*, August 1998, pp.392-403.

[14] Ming Z, Luo C, Gao W, Han R, Yang Q, Wang L, Zhan J. BDGS: A scalable Big Data generator suite in Big Data benchmarking. In *Proc. the 2013 Workshop Series on Big Data Benchmarking*, July 2014, pp.138-154.

[15] Meng X, Bradley J, Yavuz B, Sparks E, Venkataraman S, Liu D, Freeman J, Tsai D B, Amde M, Owen S, Xin D, Xin R, Franklin M J, Zadeh R, Zaharia M, Talwalkar A. MLlib: Machine learning in Apache Spark. *J. Mach. Learn. Res.*, 2016, 17: Article No. 34.

[16] Wang C, Talwar V, Schwan K, Ranganathan P. Online detection of utility cloud anomalies using metric distributions. In *Proc. the IEEE/IFIP Network Operations and Management Symposium*, April 2010, pp.96-103.

[17] Ousterhout K, Rasti R, Ratnasamy S, Shenker S, Chun B. Making sense of performance in data analytics frameworks. In *Proc. the 12th USENIX Symposium on Networked Systems Design and Implementation*, May 2015, pp.293-307.

[18] Jayathilaka H, Krintz C, Wolski R. Detecting performance anomalies in cloud platform applications. *IEEE Transactions on Cloud Computing*. doi: 10.1109/TCC.2018.2808289.

[19] Ramaswamy S, Rastogi R, Shim K. Efficient algorithms for mining outliers from large data sets. In *Proc. the 2000 ACM SIGMOD International Conference on Management of Data*, May 2000, pp.427-438.

[20] Breunig M M, Kriegel H P, Ng R T, Sander J. LOF: Identifying density-based local outliers. In *Proc. ACM SIGMOD International Conference on Management of Data*, May 2000, pp.93-104.

[21] Yu D, Sheikholeslami G, Zhang A. FindOut: Finding outliers in very large datasets. *Knowledge and Information Systems*, 2002, 4(4): 387-412.

[22] Yu L, Lan Z. A scalable, non-parametric method for detecting performance anomaly in large scale computing. *IEEE Transactions on Parallel and Distributed Systems*, 2016, 27(7): 1902-1914.

[23] Tan J, Pan X, Marinelli E, Kavulya S, Gandhi R, Narasimhan P. Kahuna: Problem diagnosis for MapReduce-based cloud computing environments. In *Proc. the IEEE/IFIP Network Operations and Management Symposium*, April 2010, pp.112-119.

[24] Pan X, Tan J, Kavulya S, Gandhi R, Narasimhan P. Ganesha: BlackBox diagnosis of MapReduce systems. *SIGMETRICS Performance Evaluation Review*, 2009, 37(3): 8-13.

[25] Gupta C, Sinha R, Zhang Y. Eagle: User profile-based anomaly detection for securing Hadoop clusters. In *Proc. the 2015 IEEE International Conference on Big Data*, October 2015, pp.1336-1343.

[26] Kasick M P, Tan J, Gandhi R, Narasimhan P. Black-box problem diagnosis in parallel file systems. In *Proc. the 8th USENIX Conference on File and Storage Technologies*, February 2010, pp.43-56.

[27] Fu X, Ren R, McKeez S A, Zhan J, Sun N. Digging deeper into cluster system logs for failure prediction and root cause diagnosis. In *Proc. IEEE International Conference on Cluster Computing*, September 2014, pp.103-112.

[28] Khan L, Awad M, Thuraisingham B. A new intrusion detection system using support vector machines and hierarchical clustering. *The VLDB Journal*, 2007, 16(4): 507-521.

[29] Lee S, Shin K G. Probabilistic diagnosis of multiprocessor systems. *ACM Computing Surveys*, 1994, 26(1): 121-139.

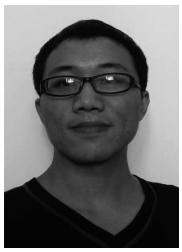
[30] Das K, Schneider J. Detecting anomalous records in categorical datasets. In *Proc. the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, August 2007, pp.220-229.

[31] Mi H, Wang H, Zhou Y, Lyu M R, Cai H. Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 2013, 24(6): 1245-1255.

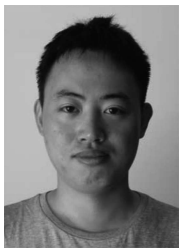
[32] Jia T, Chen P, Yang L, Li Y, Meng F, Xu J. An approach for anomaly diagnosis based on hybrid graph model with logs for distributed services. In *Proc. the 2017 IEEE International Conference on Web Services*, June 2017, pp.25-32.

[33] Ren R, Tian S, Wang L. Online anomaly detection framework for Spark systems via stage-task behavior modeling. In *Proc. the 15th ACM International Conference on Computing Frontiers*, May 2018, pp.256-259.

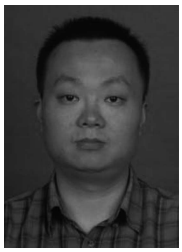
Rui Ren received her B.S. degree in computer science from the Sichuan University, Chengdu, in 2009, her M.S. degree in computer architecture from Chinese Academy of Sciences, Beijing, in 2012, and her Ph.D. degree in computer software and theory from Chinese Academy of Sciences, Beijing, in 2019. She is currently an engineer in the Institute of Computing Technology, Chinese Academy of Sciences, Beijing. Her research interests include big data, performance analysis and optimization.



Jiechao Cheng received his M.S. degree in software engineering at Wuhan University, Wuhan, in 2018, and B.S. degree at Yangtze University, Jingzhou, in 2014. He recently is a Master student at School of Computing, National University of Singapore, Singapore. His current interests focus on natural language processing and machine learning.



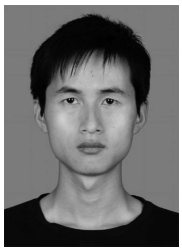
Xi-Wen He received his B.S. degree in computer science and technology from Jilin University, Changchun, in 2014, his M.S. degree in computer architecture from University of Chinese Academy of Sciences, Beijing, in 2017. He is currently an engineer in the Institute of Computing Technology, Chinese Academy of Sciences, Beijing. His research interests include big data and performance analysis.



Lei Wang received his B.S. degree in applied mathematics from Beijing University of Technology, Beijing, in 1999, and his M.S. degree in computer engineering and his Ph.D. degree in computer software and theory from the University of Chinese Academy of Sciences, Beijing, in 2006 and 2016, respectively. He is currently a senior engineer with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing. His current research interest includes benchmarking and resource management of cloud systems.

Jian-Feng Zhan received his Ph.D. degree in computer engineering from Chinese Academy of Sciences, Beijing, in 2002. He is currently a professor of computer science with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing. His current research interests include big data, distributed and parallel systems.

Wan-Ling Gao is an assistant professor in computer science at the Institute of Computing Technology, Chinese Academy of Sciences, and University of Chinese Academy of Sciences, Beijing. Her research interests focus on big data benchmark and big data analytics. She received her B.S. degree in software engineering from Huazhong University of Science and Technology, Wuhan, in 2012, and her Ph.D. degree in computer software and theory from Institute of Computing Technology, Chinese Academy of Sciences, and University of Chinese Academy of Sciences, Beijing, in 2019.



Chun-Jie Luo is a Ph.D. candidate at the Institute of Computing Technology, Chinese Academy of Sciences, Beijing. His research interests include artificial intelligence and benchmark. He received his M.S. degree in computer application technology from Chinese Academy of Sciences, Beijing, in 2012.