MIA: Metric Importance Analysis for Big Data Workload Characterization

Zhibin Yu, Wen Xiong, Lieven Eeckhout, Zhendong Bei, Avi Mendelson, and Chengzhong Xu

Abstract—Data analytics is at the foundation of both high-quality products and services in modern economies and societies. Big data workloads run on complex large-scale computing clusters, which implies significant challenges for deeply understanding and characterizing overall system performance. In general, performance is affected by many factors at multiple layers in the system stack, hence it is challenging to identify the key metrics when understanding big data workload performance. In this paper, we propose a novel workload characterization methodology using ensemble learning, called Metric Importance Analysis (MIA), to quantify the respective importance of workload metrics. By focusing on the most important metrics, MIA reduces the complexity of the analysis without losing information. Moreover, we develop the MIA-based Kiviat Plot (MKP) and Benchmark Similarity Matrix (BSM) which provide more insightful information than the traditional linkage clustering based dendrogram to visualize program behavior (dis)similarity. To demonstrate the applicability of MIA, we use it to characterize three big data benchmark suites: HiBench, CloudRank-D and SZTS. The results show that MIA is able to characterize complex big data workloads in a simple, intuitive manner, and reveal interesting insights. Moreover, through a case study, we demonstrate that tuning the configuration parameters related to the important metrics found by MIA results in higher performance improvements than through tuning the parameters related to the less important ones.

Index Terms—Big data, benchmarking, workload characterization, performance measurement, MapReduce/hadoop

1 Introduction

As our planet is becoming increasingly digitally instrumented and connected, we notice that the data volume of the world has experienced a steep increase over the past five years, which is predicted to increase even faster in the future. As a result, big data has become an overnight buzzword all over the world. Governments, companies, and other organizations embrace big data enthusiastically because they believe precious deposits are hidden in big data. To mine the precious value, big data workloads need to run on complex large-scale computing clusters such as cloud platforms with emerging big data frameworks such as MapReduce/Hadoop [28], [29] and Hive [6].

It is notoriously challenging, yet crucially needed, to understand and characterize the performance of large-scale compute clusters to optimize performance. The challenges in characterizing big data workloads and systems come from the large-scale distributed nature of the system and the multi-layered software stack. On the one hand, big data hardware platforms typically consist of a couple thousands of servers. To deeply understand such a platform, a hierarchical characterization must be done at the node level (including detailed characterization at the processor level covering caches, branch predictor, TLBs, etc.), as well as at the system level including the interconnection network and storage devices.

On the other hand, software in big data systems not only contains a large number of application-level jobs but also consists of a multi-layered system software stack, including a distributed file system (e.g., HDFS), hypervisor, operating system, and a big data analysis framework such as Hadoop. As is the case for hardware, software characterization also needs to be performed at multiple levels including the job, stage, and task levels. Moreover, software and hardware of big data systems interact with each other tightly, indicating that we need to consider hardware and software metrics at multiple levels in the system stack together when we want to gain deep understanding of the overall system. As a result, typically tens to hundreds of metrics are involved.

More formally, performance can be described as follows:

$$perf = f(hm_1, \ldots, hm_i, \ldots, hm_k, sm_1, \ldots, sm_j, \ldots, sm_n), \quad (1)$$

with $perf$ the performance of an application; hm_i and sm_j the i-th hardware and j-th software metric, respectively; and f the function that relates performance to the input metrics. (k and n are the numbers of available hardware and software metrics, respectively.) In big data systems, performance can be quantified using a range of metrics including data processing speed (DPS), throughput, latency, or any other metric such as instructions per cycle (IPC), depending on an end user's requirements. Obviously, it is extremely difficult to define a analytical model (e.g., formula) for function f because there are at least

- *Z. Yu, W. Xiong, Z. Bei, and C. Xu are with the Cloud Computing Center, Shenzhen Institutes of Advanced Technology, Chinese Academy of Science, Shenzhen 518055, China. E-mail: {zb.yu, wen.xiong, zd.bei, cz.xu}@siat.ac.cn.*
- *L. Eeckhout is with the Ghent University, Ghent 9000 Belgium. E-mail: lieven.eeckhout@ugent.be.*
- *A. Mendelson is with the Technion-Israel Institute of Technology, Haifa 3200003, Israel. E-mail: avi.mendelson@tce.technion.ac.il.*

Manuscript received 26 Apr. 2016; revised 28 Aug. 2017; accepted 5 Sept. 2017. Date of publication 4 Oct. 2017; date of current version 11 May 2018. (Corresponding author: Wen Xiong.)
Recommended for acceptance by C. Carothers.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2017.2758781

several hundreds of input parameters that interact with each other in various complex ways.

One possible alternative is to construct empirical data models for f using machine learning. However, it is difficult to construct accurate data models at low cost (e.g., the time used to collect training data), as we will quantify in this paper. From a practical viewpoint it is of critical importance to know which metrics are important and should be considered when characterizing and optimizing big data workloads. Knowing the key metrics affecting performance helps gear the performance analyst's attention to fruitful optimizations, and avoids spending time on non-productive analyses. However, without in-depth understanding of the software and hardware at hand of a big data system, identifying the key metrics is extremely challenging.

In this paper, we propose Metric Importance Analysis (MIA), a novel workload characterization methodology that uses ensemble learning to quantify the relative importance of the metrics or characteristics of a big data system with respect to system-level performance (DPS: data processing per second) and node-level performance (IPC: instructions executed per cycle). MIA ranks the metrics according to their relative importance. As a result, one can focus on the important metrics while ignoring the less important ones. This provides a way to comprehensively characterize big data systems by focusing on a limited number of important metrics. Although several characterization studies have been done on big data systems yielding good insights [1], [2], [7], [8], [10], [11], [27], these studies considered a few metrics only based on the experimenter's intuition. In contrast, MIA provides a way to investigate metrics in a systematic way, quantify their importance, and select the important ones for further analysis. In other words, MIA provides a systematic way to fully understand the performance of big data systems in an intuitive way at low effort.

We further propose the MIA-based Kiviat Plot (MKP) by taking the n most important metrics in a descending order as the axes (in clockwise direction) of a Kiviat plot—n is the minimum number of metrics for which the cumulative importance exceeds for example 80 percent. As such, behavioral (dis)similarity between benchmarks can be easily observed by comparing the corresponding Kiviat plots. If two plots are (dis)similar, the two corresponding benchmarks exhibit (dis)similar behavior. Although the traditional Linkage Clustering-based Dendrogram (LCD) can visualize behavioral (dis)similarity between benchmarks as well, the advantages of MKP over LCD are threefold. (1) MKP facilitates a more direct view on the (dis)similarity between benchmarks. (2) MKP provides more information to show the (dis)similarity between benchmarks because it uses the n most important metrics whereas LCD only employs a 'summary' distance between the corresponding metrics vectors. (3) MKP does not visualize all metrics but only the n most important ones, which reduces noise while guaranteeing the accuracy of the similarity characterization.

In addition, we propose the Benchmark Similarity Matrix (BSM) to visually summarize the (dis)similarity among a set of benchmarks. One element of the BSM represents the (dis)similarity between two benchmarks and the (dis)similarity is quantified by the Manhattan distance between the two corresponding vectors of two benchmarks. A vector consists of the important metrics of its corresponding benchmark. The (dis)similarity is visualized using blue scale: the darker the point is, the more similar the two benchmarks are; the lighter the point is, the more dissimilar the two benchmarks are.

We use MIA to perform a cross-layer workload characterization of three big data benchmark suites: HiBench [1], CloudRank-D [2], and SZTS [27]. The results show that MIA, MKP, and BSM are indeed useful to characterize the behavior of complex big data systems in an intuitive way while revealing interesting insights. From a system's perspective, we find that the miss rate of the last-level cache is the most important factor for IPC, whereas the number of temporary read/write operations is the most important metric for DPS. From a methodology perspective, we find it is necessary to perform MIA using metrics across multiple layers because a metric at one layer may have impact on a performance metric at another layer. From a benchmark perspective, we find that the programs from the SZTS benchmark suite exhibit significantly different behavior compared to the other benchmark suites.

As a case study to demonstrate the usefulness of MIA, we leverage the results from MIA to steer performance optimization. MIA points out that the amount of temporary data (TMI) is significantly more important than the ratio of *map* function time to *reduce* function time (TMRF). We then adjust the value of a configuration parameter that is tightly related to TMI and the number of reducers that is tightly related to TMRF to optimize performance for `terasort`. The results indeed show that adjusting the former parameter achieves significantly higher performance than the latter one. This indicates that MIA can help one optimize a program more efficiently by focusing on the more important metrics.

Note that although we consider Hadoop in this study, this does not limit the general applicability of MIA to other big data frameworks such as Spark [47]. Furthermore, we believe MIA is generally applicable on any other big data systems because it does not rely on specific properties of the big data system at hand.

In summary, we make the following contributions:

- We propose a novel big data workload characterization methodology using ensemble learning, called Metric Importance Analysis, which quantifies the relative importance of workload metrics.
- We propose the MIA-based Kiviat Plot and Benchmark Similarity Matrix to visualize the performance behavior (dis)similarity among big data workloads in an intuitive way.
- We employ MIA to characterize three big data benchmark suites: HiBench, CloudRank-D, and SZTS. We derive several interesting insights regarding system performance; methodology for characterizing big data workloads; and benchmark characteristics.
- We demonstrate through a case study that MIA can help optimize programs more efficiently by tuning the most important configuration parameters first.

The rest of this paper is organized as follows. Section 2 describes the Hadoop framework we use in this paper. Section 3 describes the proposed MIA methodology. Section 4 introduces our experimental setup. Section 5

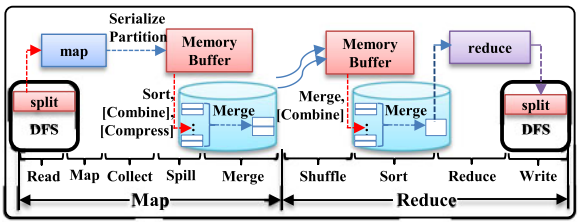


Fig. 1. The stages and phases of a MapReduce program.

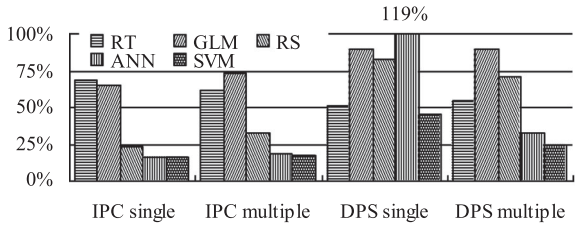


Fig. 2. The error of performance models built by RT (regression tree), GLM (generalized linear module), RS (response surface), ANN (artificial neural network), and SVM (supported vector machine).

presents our characterization results and analysis. Section 6 discusses related work, and finally, Section 7 concludes the paper.

2 Background and Motivation

2.1 MapReduce/Hadoop

MapReduce/Hadoop is a highly scalable programming model/framework for processing and generating large data sets formatted in key/value pairs. Users only need to write *map* and *reduce* functions, and the rest is handled by the Hadoop runtime. Large input data sets are split into small blocks by the Hadoop Distributed File System (HDFS). As shown in Fig. 1, the execution of a Hadoop program can be divided into *map* and *reduce* stages. In the *map* stage, each *map* task reads a small block and processes it. When *map* tasks complete, the outputs—also known as intermediate files—are copied to the *reduce* nodes. At the *reduce* stage, *reduce* tasks fetch the key/value pairs from the output files of the *map* stage, which they then sort, merge, and process, to produce the final output. The *map* stage itself can be further divided into *read*, *map*, *collect*, *spill*, and *merge* phases. Similarly, the *reduce* stage can be divided into *shuffle*, *merge*, *reduce*, and *write* phases.

2.2 Limitations in Prior Modeling Work

As argued above, we need tens to hundreds of metrics to comprehensively understand big data workload behavior. Characterizing workloads typically involves building performance models as functions of these metrics. Statistical reasoning algorithms such as regression trees (RT) [42], [43], [44] and response surfaces (RS) [45], as well as machine learning techniques such as artificial neural networks (ANN) [46] and support vector machines (SVM) [52] have been used to build performance models for computer systems. These models can be made very accurate if given a large number (several hundreds or thousands) of training examples. Unfortunately, collecting this large a number of training examples is non-trivial for a big data system.

The errors for the above models when given 106 training examples are reported in Fig. 2. (Section 4 provides details regarding the experimental setup.) 'IPC/DPS single' means that the metrics used as input to the performance models are taken from a single respective layer (either the node level for IPC and the system level for DPS), whereas 'IPC/DPS multiple' indicates that the metrics are taken from both layers. As can be seen, the average error exceeds 19 percent in all cases. This clearly indicates that these widely used statistical reasoning and machine learning techniques are inaccurate when given a limited number of training examples in the context of a big data system. Building a methodology on top of such an inaccurate technique to identify the most important workload characteristics is flawed. We hence need a more advanced modeling technique that can achieve high accuracy with a relatively small training set. We employ ensemble learning for this purpose as described in the next section.

3 MIA Methodology

3.1 MIA Workflow

Fig. 3 illustrates the MIA workflow which consists of the following four components: profiling, storing, modeling, and applications. MIA profiling measures performance characteristics at multiple levels of a Hadoop workload. MIA storing refers to storing the profiling results in a performance database, see step ①. We employ different tables to store the metrics from different layers in the database. For example, a table named `tbl_micro` stores processor-level micro-architecture metrics such as the L1 data cache miss rate, branch misprediction rate, etc.; the `tbl_sys` table stores the metrics at the system level. MIA modeling leverages an ensemble learning algorithm, called stochastic gradient boosted regression tree (SGBRT), to create performance models using the performance database as training data, see step ②. Based on these models, MIA quantitatively analyzes the importance of the metrics in terms of a certain performance metric such as IPC or DPS, as step ③ illustrates. The last component of the MIA workflow shows two cases using the importance metrics, see step ④. The first one is to employ the n most important metrics (see Section 5 for its determination) as the axes of a Kiviat plot (MKP). The axes in clockwise direction represent the important metrics in descending order of importance. As such, MKP can be used to visualize the behavior of a workload. Moreover, by comparing the (dis)similarity between two MKPs, we can easily observe the behavioral (dis)similarity between the two corresponding workloads. The second use case is to use the values of the important metrics of a benchmark as the elements of a vector and subsequently construct a Benchmark Similarity Matrix by employing the Manhattan distance of two vectors as an element in the matrix. As such, BSM can visually summarize the (dis)similarity among benchmarks.

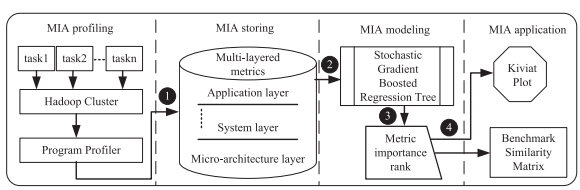


Fig. 3. The MIA workflow.

TABLE 1
Job-Level Metrics

Metric	Description
DPS	data processing speed
MOI	(map output)/(map input)
SMI	(shuffle data)/(map input)
TMRS	(map stage time)/(reduce stage time)
TMRF	(map function time)/(reduce function time)
ROI	(reduce output)/(reduce input)
ROMI	(reduce output)/(map input)
TMI	(temporal write data)/(map input)
TRAMI	(intra-node transmitted data)/(map input)
TERMI	(inter-node transmitted data)/(map input)

We treat DPS as the dependent metric and the others as independent metrics.

TABLE 2
Microarchitecture-Level Metrics

Metric	Description
IPC	instructions/cycle
L1ICMPKI	no. L1 icache misses/1K insns
L2ICMPKI	no. L2 icache misses/1K insns
LLCMPKI	no. last-level cache misses/1K insns
BRPKI	no. branches/1K insns
BRMPKI	no. branch misses/1K insns
OCBW	off-chip bandwidth utilization
FSPKC	no. instruction fetch stall cycles/1K cycles
RSPKC	no. resource related stall cycles/1K cycles
DTLBPKI	no. D-TLB load misses/1K insns

We treat IPC as the dependent metric and the others as independent metrics.

The key component of the MIA methodology is the underlying data model because it determines the accuracy of the MIA results and the cost for obtaining these results. Inaccurate performance models may jeopardize the accuracy of the metric importance analysis. On the other hand, high accuracy is likely to incur a high cost in terms of profiling to collect many training examples to construct the performance models. Moreover, many training examples may lead to over-fitting the models. As argued in Section 2.2, widely used statistical reasoning and machine learning techniques fail to be accurate in the context of a big data system.

In this paper, we therefore opt for stochastic gradient boosted regression tree [23] to perform MIA because it is shown to be more accurate and incur lower cost compared to the aforementioned techniques. SGBRT combines multiple single models (regression trees) to create a final prediction or classification whereas the traditional algorithms only use a single model as their final model. Combining multiple models not only improves accuracy, it also makes the overall model more robust to over-fitting.

3.2 Metrics

To fully understand the behavior of a big data system workload, we argue that we need to collect as many metrics as possible from multiple layers in the system stack, including the micro-architecture and job level. Quantifying micro-architecture level metrics helps understanding node (processor) performance, while the job-level metrics provide insight into overall system performance. Although metrics from other levels such as the Operating System (OS) may also be included, we focus on job- and microarchitecture-level metrics in this study.

3.2.1 Job-Level Metrics

The metrics we consider at the job level are enumerated in Table 1; these metrics are chosen to provide a fairly broad view on the amount of input and output data, processing speed and communication at the job level. *Data Processing Speed* measures how fast a program processes data, and is defined as the amount of data processed divided by the execution time of a program. It provides a system view on performance, and is a higher-is-better metric.

To quantify how the amount of data changes before and after the *map* stage, we define *Map Output/Input ratio (MOI)* as the ratio of the *map* output data size to its input. *Shuffle/Map Input ratio (SMI)* is defined as the ratio of the amount of data processed by the *shuffle* operation to that processed by the *map* operation. *Time Map/Reduce Stage ratio (TMRS)* quantifies the ratio of the execution time of the *map* stage to that of the *reduce* stage. *Time Map/Reduce Function ratio (TMRF)* is defined similarly for the *map* and *reduce* functions. (Note that the *map* stage takes longer than the *map* function since the *map* stage includes operations such as *spill* and *merge*, as previously mentioned. The same applies to the *reduce* stage/function.) *Reduce Output/Input ratio (ROI)* is the ratio of the amount of data of the *reduce* output to that of the *reduce* input. *Reduce Output to Map Input ratio (ROMI)* is defined as the ratio of the amount of data of the *reduce* output to that of the *map* input. *Temporal to Map Input ratio (TMI)* is the ratio of the amount of data temporally written to the local file system to that of the *map* input. *Intra-node to Map Input (TRAMI)* is the ratio of the amount of data transmitted between processes within a node to that of the *map* input. *Inter-node to Map Input (TERMI)* is the ratio of the amount of data transmitted between nodes to that of the *map* input.

3.2.2 Microarchitecture-Level Metrics

Table 2 shows the metrics we use to characterize our benchmarks at the microarchitecture level; these metrics, collectively, provide a good view on the performance of individual nodes. We collect these metrics on each node using hardware performance counters, and then compute the average across all eight nodes. (See Section 4).

Instructions Per Cycle computes the number of instructions executed per cycle, and is a reliable metric for node-level performance (higher is better). Although IPC is not a perfect measure, it closely correlates with node performance. *L1ICMPKI* and *L2ICMPKI* quantifies the number of L1 and L2 instruction cache misses, respectively, per thousand instructions, and is a measure for the instruction footprint and code locality (lower is better). *LLCMPKI* quantifies the number of last-level cache (LLC) misses per thousand instructions, and is a metric for the data footprint and locality (lower is better). We do not consider L1 and L2 data cache misses as most of these latencies can be hidden by out-of-order execution (as supported by the processors considered in this paper). *BRPKI* is the number of branches per one thousand instructions, and *BRMPKI* is the number of branch mispredictions per one thousand instructions (lower is better). *OCBW* is the off-chip bandwidth utilization per-core measured in bytes per second, and is calculated as

$$(64 \times 2.4 \times 10^9) \times llcm/clkuh,$$

with 64 the number of bytes fetched, 2.4 GHz the frequency of the processor, *llcm* the number of LLC misses and *clkuh* the number of unhalted clock cycles. *IFSKC* and *RSKC* are defined as the numbers of stall cycles due to instruction fetch stalls and other resource stalls, respectively. *DTLBPKI* quantifies the number of D-TLB misses per 1K instructions.

3.3 Identifying Important Metrics

The total amount of data measured and collected during this characterization is substantial: 10 job-level metrics and 10 microarchitecture-level metrics for all 133 benchmark-input pairs. (Note that some programs contain several jobs and we treat each job as a single program.) In addition, each program has several input data sets. (See Section 4 for further details.) This yields 2,660 data values in total per server. Considering the eight server nodes in our setup, this amounts to a total of 21,280 data values.

Analyzing this big a data set is challenging, hence we propose a novel workload characterization methodology, called Metric Importance Analysis, which identifies the most significant workload characteristics. MIA starts off by identifying a *dependent* variable that is the primary performance metric; this is IPC at the microarchitecture level, and DPS at the job level. MIA then ranks the *independent* variables (the other workload characteristics) by their importance with respect to the dependent variable. We employ ensemble learning to infer this ranking, namely Stochastic Gradient Boosted Regression Trees [23], which requires relatively few training examples to construct an accurate empirical model.

SGBRT is an ensemble model that 'boosts' the prediction accuracy of traditional machine-learning algorithms by leveraging the combination of the predictions produced by multiple independent regression trees to perform the final prediction. Therefore, it is expected that the models built by SGBRT are more accurate and robust than traditional machine-learning algorithms, which typically rely on a single model. SGBRT works well for a range of problem domains. We refer the interested reader to [23] for more background information regarding SGBRT.

Now we focus on describing how to build performance models using SGBRT. First, we collect the values of DPS, IPC, and all the metrics described in Tables 1 and 2 when we run each program-input pair. Second, we build a DPS vector and an IPC vector using the data for each program-input pair as follows:

$$DPSV_i = \{DPS_i, m_{i1}, \ldots, m_{ij}, \ldots, m_{in}\}, i = 1, \ldots, N \quad (2)$$

$$IPCV_i = \{IPC_i, m_{i1}, \ldots, m_{ij}, \ldots, m_{ik}\}, i = 1, \ldots, N, \quad (3)$$

with $DPSV_i$ and $IPCV_i$ the DPS vector and IPC vector respectively for the i-th program-input pair; DPS_i and IPC_i the data processing speed and instruction per cycle, respectively, for the i-th program-input pair; m_{ij} the value of the j-th metric for the i-th program-input pair; n the number of metrics selected for the DPS vector; k the number of metrics selected for the IPC vector; and N the number of program-input pairs.

The DPS and IPC vectors for the N program-input pairs form the DPS matrix and IPC matrix, respectively

$$DPSM = \{DPSV_1, DPSV_2, \ldots DPSV_i, \ldots, DPSV_N\} \quad (4)$$

$$IPCM = \{IPCV_1, IPCV_2, \ldots IPCV_i, \ldots, IPCV_N\}. \quad (5)$$

The $DPSM$ and $IPCM$ matrices are used as input to SGBRT to train a DPS model and a IPC model, respectively

$$DPS = f_{DPS}(m_1, \ldots m_j, \ldots, m_n) \quad (6)$$

$$IPC = f_{IPC}(m_1, \ldots m_j, \ldots, m_k), \quad (7)$$

with m_j the metrics shown in Tables 1 and 2.

Once a performance model is constructed for the dependent metric, IPC and DPS in our case, one can start ranking the independent metrics. We call this novel method, Metric Importance Analysis, which is done as follows. For a single tree T, one can use $I_j^2(T)$ as the measure of importance for each independent metric x_j (e.g., *MOI—(map output)/(map input)*), which is based on the number of times x_j was selected for splitting the tree weighted by the squared improvement to the model as a result of each of those splits [24]. We calculate it as follows:

$$I_j^2(T) = nt \cdot \sum_{i=1}^{nt} P^2(i), \quad (8)$$

with nt the number of times x_j is used to split tree T, and $P^2(i)$ the squared improvement to the tree model for the i-th split. For example, $P(i)$ may be the relative IPC error $(IPC_i - IPC_{i-1})/IPC_{i-1}$ after the i-th split. If x_j is used as a splitter in M trees of the ensemble model, the importance of x_j to the model equals

$$I_j^2 = \frac{1}{M} \sum_{m=1}^{M} I_j^2(T_m). \quad (9)$$

Once the importance factors are computed for each independent metric, we normalize the importance so that the sum across all independent metrics adds up to 100 percent. The higher the (normalized) importance, the more significant the impact is of the respective independent metric.

In this work, we consider node-level performance (IPC) and system-level performance (DPS) as the dependent metrics, and we build separate models for each. An important question then arises which independent metrics to consider for each model. Should we only consider node-level characteristics for the node-level performance model, or should we also consider job-level metrics? And vice versa, should we only consider job-level metrics in the job-level performance model, or should we also consider node-level characteristics? Interestingly, we find that the node-level performance model is more accurate when considering both node-level and job-level characteristics as the independent characteristics. The same applies to the system-level performance model. (More details are provided in Section 5.) This indicates that there is a strong interplay between the microarchitecture- and job-level characteristics. We therefore use both in MIA.

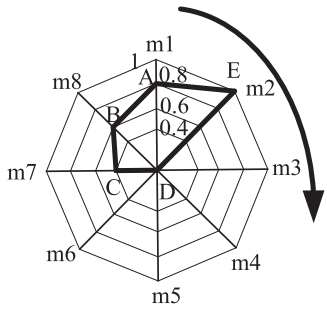


Fig. 4. Overview of MIA-based Kiviat plot. m1, m2, ..., m8 are the 8 important metrics of a Hadoop workload in terms of IPC or DPS (note that we use n = 8 as example to illustrate MKP). The relative importance for each metric varies between 0 and 1.

3.4 MIA-Based Kiviat Plot

Visualization can greatly facilitate workload characterization when analyzing program behavior. Eeckhout et al. [17] proposed linkage clustering and dendrograms to visualize (dis)similarity between workloads based on their respective behavior vectors. A large linkage distance between two workloads indicates dissimilar behavior, whereas a short distance indicates similar behavior. While this is an intuitive way to visualize workload (dis)similarity, it does not visualize why workloads are (dis)similar, because the linkage distance is based on a 'summary' distance across the behavioral vectors characterizing the respective workloads. To address this issue, we propose the MIA-based Kiviat Plot to better visualize the (dis)similarity between benchmarks. MKP is used to reveal detailed information regarding the (dis)similarity between two benchmarks.

MKP is constructed as follows. MIA first quantifies the importance of each metric and sorts all metrics in a descending order according to their importance. We then choose the first n most important metrics to summarize program behavior. n is the minimum number of metrics for which the cumulative importance exceeds for example 80 percent. Finally, we draw a Kiviat plot for each benchmark using the n metrics as its axes; the most important metric is shown at the 12 o'clock position, and the importance of metrics decreases following the clockwise direction, as shown in Fig. 4.

Thus far, we have determined the number and the order of axes of MKPs for all benchmarks but have not set the value of each axis yet. (Note that the MKPs for all benchmarks have the same axes in the same order but the same axis for different benchmarks might have different values.) One option may be to represent the raw value of the corresponding metric. However, the values may be very different across metrics, making it hard to visualize and compare MKPs. We therefore normalize the value for each metric as follows:

$$M_{nor} = \frac{M_{ori} - M_{min}}{M_{max} - M_{min}}, \tag{10}$$

with M_{nor} the normalized value of metric M; M_{ori} the original value of M; M_{max} and M_{min} the maximum and minimum value of M across all benchmarks. Hence, normalized values range between 0 and 1, which simplifies the analysis. For example, the polygon A-B-C-D-E in Fig. 4 visualizes the behavior of a given benchmark with values 0.8, 0.6, 0.4 and 1 for metrics $m1$, $m8$, $m7$ and $m2$, respectively.

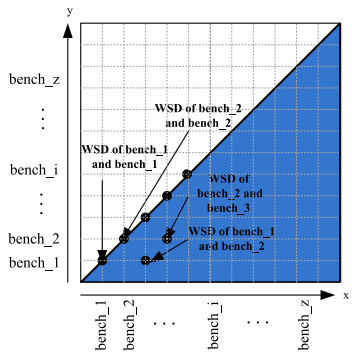


Fig. 5. An overview of the benchmark similarity matrix.

MKP has three advantages over the dendrogram provided by linkage clustering [17]. (1) It provides a more intuitive and informed view of the (dis)similarity between benchmarks by comparing the corresponding shapes and their direction. (2) MKP reveals more information related to the (dis)similarity between benchmarks because it compares at least n important metrics. (3) MKP visualizes the most important n metrics, which reduces noises while guaranteeing the accuracy of the characterization. Previously proposed Kiviat plots [41] select metrics based on the experimenter's intuition, which may affect the validity and accuracy of the characterization, i.e., some of these metrics may have no or limited impact on performance, which would mislead a performance analyst.

3.5 Benchmark Similarity Matrix

Having described MKP, we can now define the Benchmark Similarity Matrix which eases the visualization of (dis)similarity among a set of benchmarks. The BSM is based on the notion of the Weighted Summary Distance (WSD) between two benchmarks. We define an importance vector v to represent the program behavior of each benchmark as follows:

$$v = \{wm_1, wm_2, \ldots, wm_i, \ldots, wm_k\}, \tag{11}$$

with wm_i the value of the i-th important metric, and k the total number of metrics used to describe the behavior of a benchmark, e.g., the n important metrics selected for MKP. The WSD between two benchmarks is then calculated as the Manhattan distance between the two corresponding importance vectors v_p and v_q

$$ManhattanDist(v_p, v_q) = \sum_{i=1}^{k} |wm_i^p - wm_i^q|, \tag{12}$$

with wm_i^p the value of the i-th important metric of benchmark p, and wm_i^q the value of the i-th important metric of benchmark q. Note that we choose Manhattan distance because it has the advantage that it weights more heavily differences in each dimension compared to the Euclidean distance, i.e., being closer in the x-dimension does not get you any closer in the y-dimension.

The BSM is a novel and intuitive tool to visually summarize the (dis)similarity between a set of benchmarks. BSMs are triangles (bottom half of a matrix), in which the blue scale of a point represents the similarity between two benchmarks, see Fig. 5 for an example. A point (x, y) in the BSM $(x \geq y)$ represents the (dis)similarity between benchmarks x

TABLE 3
Processor Configuration

Item	Value
CPU type	Intel Xeon E5620
# cores	8 cores, 2.4 GHz
# threads	16 threads
# sockets	2
I-TLB	4-way, 64 entries
D-TLB	4-way, 64 entries
L2 TLB	4-way, 512 entries
L1 DCache	32 KB, 8-way, 64 bytes/line
L1 ICache	32 KB, 4-way, 64 bytes/line
L2 Cache	256 KB, 8-way, 64 bytes/line
L3 Cache	12 MB, 16-way, 64 bytes/line

and y. (Dis)similarity is quantified by the WSD between the two corresponding vectors of the two benchmarks. The blue scale of a point (x, y) denotes (dis)similarity: the darker the point, the more similar the benchmarks are; the lighter the point, the more dissimilar the benchmarks are.

4 Experimental Setup

4.1 Hardware Platform

We use a Hadoop cluster consisting of one master (control) node and eight slave (data) nodes. All the nodes are equipped with two 1 GB/s Ethernet cards and are connected through two Ethernet switches. The A switch is used for global clock synchronization whereas the B switch is used to route Hadoop communication. Each node has three 2 TB disks, 16 GB memory, and two Intel Xeon E5620 multicore (Westmere) processors. The detailed configuration of each processor is shown in Table 3. Each processor contains 8 cores, with each core being a 4-wide superscalar architecture. The operating system running on each node is Ubuntu 12.04, kernel version 3.2.0. The versions of Hadoop and JDK are 1.0.3 and 1.7.0, respectively.

4.2 Measurement Tools

We measure the microarchitecture-level characteristics using `oprofile` v0.98 which reads the hardware performance counters through sampling. We use `sysstat` v9.06 to collect system-level metrics. In addition, we read the log file of the Hadoop framework every 5 seconds to observe the Hadoop job features. Finally, we also read */proc/net/dev* every 5 seconds to obtain runtime characteristics of the network.

4.3 Benchmarks

The benchmarks considered in this paper are summarized in Table 4. Next to the SZTS benchmarks described in [27], we also consider benchmarks from HiBench [1] and Cloudrank-D 1.0 [2]. The inputs for the latter benchmark suites are synthetically generated. The SZTS benchmarks on the other hand come with real-life inputs. Note that a Hadoop program may consist of multiple jobs. Each job typically consists of a *map* task and a *reduce* task. From this point of view, a Hadoop job is actually a small Hadoop program. Hence, we treat different jobs of the same program as different benchmarks. The *no. jobs* column in Table 4 lists the number of jobs per program.

TABLE 4
Benchmarks and their Input Data Sets

Benchmark	Suite	No. jobs	Input data set (GB)
terasort	HiBench	1	100, 400, . . ., 1000
sort	HiBench	1	20, 40, . . ., 400
wordcount	HiBench	1	20, 40, . . ., 400, 500
kmeans	HiBench	2	26, 50, 80, 107, 201
pagerank	HiBench	2	12, 24, 53, 80, 109
hive-aggre	HiBench	1	23, 61, 83, 100, 140
hive-join	HiBench	3	23, 61, 83, 100, 140
grep	CR-D	1	50, 100, 160, 200, 300
hmm	CR-D	1	50, 100, 160, 200, 300
nbayes	CR-D	4	10, 30, 50, 80, 100
sztod	SZTS	1	20, 50, 100, 160, 200
hotregion	SZTS	1	50, 100, 160, 200, 300
mapm	SZTS	1	2, 4, 8, 12, 16
hotspot	SZTS	2	50, 100, 160, 200, 300
secsort	SZTS	2	50, 100, 160, 200, 300

4.4 Generating Training Samples

We generate the training examples as follows. We run all program-input pairs on our Hadoop cluster and collect the values for all metrics shown in Tables 1 and 2. These values are stored in the DPSV and IPCV vectors from Equations (2) and (3), respectively. This is done for all N program-input pairs, with $N = 133$. Out of these 133 DPSVs, we sample 80 percent training examples which we use to construct the DPS data model; the remaining 20 percent is used for evaluation purposes (i.e., cross-validation setup in which the training set and evaluation set are disjoint). The same is done for the IPC model.

We use the R language and libraries to build our RT, GLM, RS, ANN, SVM, and SGBRT models. The source code for RT is adapted from [43]. The R libraries for GLM, RS, ANN, and SVM are 'stats', 'rsm', 'nnet', and 'e1071', respectively. The ANN is a feed-forward single-hidden-layer neural network. Although there exist other ANN architectures that can produce more accurate results, they need (much) more training examples and in turn much longer time to collect the training data. For SGBRT, we develop our own implementation by leveraging the 'gbm' library.

5 Results and Analysis

5.1 Model Accuracy

We first evaluate the accuracy of the SGBRT model and compare it against the SVM and ANN models which are the most accurate ones shown in Section 2.2. In fact, we build two prediction models, one for IPC as shown in Equation (7) and one for DPS as shown in Equation (6). For the IPC model, we consider two scenarios, one in which we only consider microarchitecture-level metrics as its input (the m_j's in Equation (7) can only be microarchitecture-level metrics), and one in which we consider both microarchitecture and job-level metrics; likewise for the DPS model. We employ cross-validation and consider different data sets for training versus evaluation, as mentioned before.

The error is reported in Fig. 6. The error is the lowest when metrics are considered from both the microarchitecture and job levels for SGBRT. In particular, the average error drops from 10.6 to 4.3 percent for the IPC model when

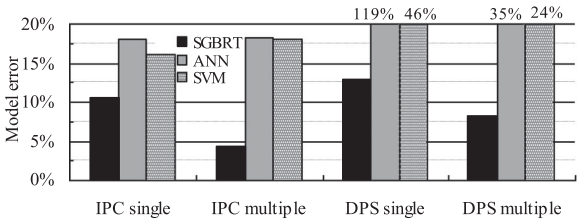


Fig. 6. Average modeling error for the IPC and DPS models when considering metrics at the respective level only ('single') versus considering metrics at both the microarchitecture and job levels ('multiple').

considering metrics at both levels as opposed to only considering the microarchitecture-level metrics. Similarly, the average error drops from 13 to 8.2 percent for the DPS model when considering metrics at both levels as opposed to job-level metrics only. The underlying reason is that there exists correlation between the job and microarchitecture level metrics; we analyze these correlations in more detail in subsequent sections.

In contrast, the errors of models built by ANN and SVM using the same training data as those of SGBRT models are substantially higher. Even when considering both job-level and microarchitecture-level metrics, the average error of the ANN models equals 18.3 and 77 percent for IPC and DPS, respectively. Likewise, the average IPC and DPS error of the SVM models is 18 and 35 percent, respectively.

5.2 MIA Results

Having shown the accuracy of the SGBRT model, we now move towards the MIA analysis. We first present the MIA results for predicting IPC and then for DPS.

5.2.1 Metric Importance for IPC

Fig. 7 shows the MIA analysis when considering both job- and microarchitecture-level metrics for constructing an IPC model. LLC-MPKI is the most important factor in terms of IPC with an importance of 15 percent. Interestingly, the second most important metric is TMRF, or the ratio between the time spent in the map versus reduce functions. The next five metrics are all microarchitecture-level metrics. The fact that TMRF is the second most important metric for the IPC model confirms that there exists correlation between job-level characteristics and the observed microarchitecture-level behavior. The fact that TMRF is found to correlate with IPC can be explained as follows. Generally speaking, the map function is in charge of reading and processing

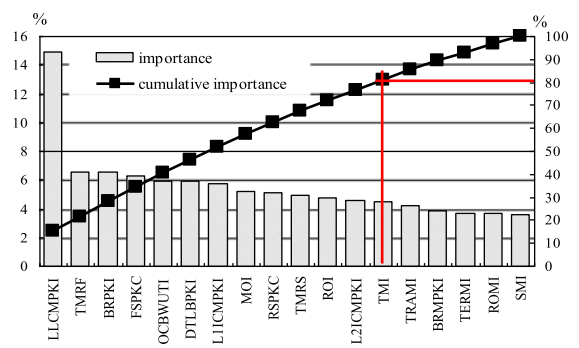


Fig. 7. Importance of the microarchitecture- and job-level metrics on IPC.

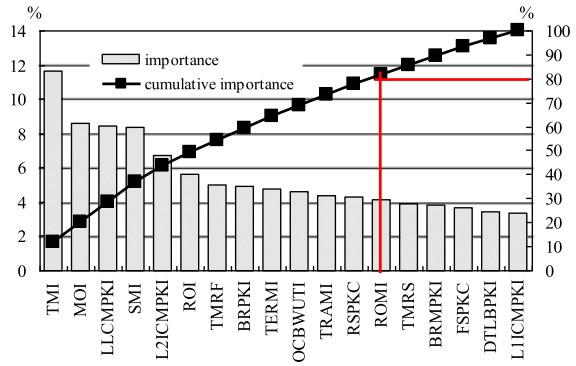


Fig. 8. Importance of the microarchitecture- and job-level metrics on DPS.

data while the reduce function shuffles and processes the data. If the reduce function is compute-intensive and needs to perform significant processing compared to the map function, this will have its impact on node-level performance, affecting IPC.

5.2.2 Metric Importance for DPS

Fig. 8 reports a similar analysis for the DPS model. TMI is the most important factor for DPS with an importance equal to 11.8 percent. The next important metrics are MOI, LLC-MPKI, and SMI, which have almost the same importance. It is interesting to note that LLC-MPKI, which is an important microarchitecture-level metric for predicting IPC, is also important for predicting DPS. This suggests that LLC-MPKI significantly affects single-node IPC, which in its turn, affects the overall DPS throughput of a node, which makes intuitive sense.

Note that metric importance might or might not change when cluster size changes, depending on whether particular components get saturated. For example, consider a workload that consumes all the available memory bandwidth in a cluster of 16 servers but other resources are plenty. Changing the cluster size to 32 is likely to still saturate memory bandwidth, hence the relative metric importance may be unaffected. In other cases where the performance bottleneck changes with changing cluster size, the relative metric importance may change.

5.3 Program Similarity Analysis

Having identified the n most important metrics for predicting IPC and DPS, we now characterize the three big data benchmark suites in terms of these metrics using MKP and BSM visualization. We start with the Kiviat plots, focusing on IPC first, followed by DPS analysis. Throughout this section, we consider $n = 13$ metrics accounting for 80 percent of the cumulative importance, see also Figs. 7 and 8.

5.3.1 MIA Kiviat Plots for IPC

We now employ MKP to investigate how the Hadoop benchmarks considered in this study differ from each other in terms of IPC, see Fig. 11. (The SZTS benchmarks are shown in shaded sub-figures). There are a number of interesting observations to be made here. For one, similar MKP results in similar IPC. For example, the MKPs are quite similar to each other for `pagerank-s1`, `hive-aggre` and `hive-join-s1`. Their IPC is also very similar,

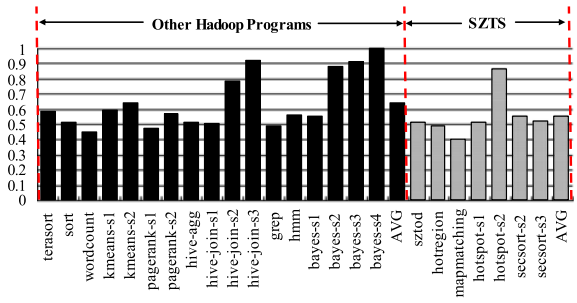


Fig. 9. IPC for SZTS and the other Hadoop benchmarks (HiBench and CloudRank-D).

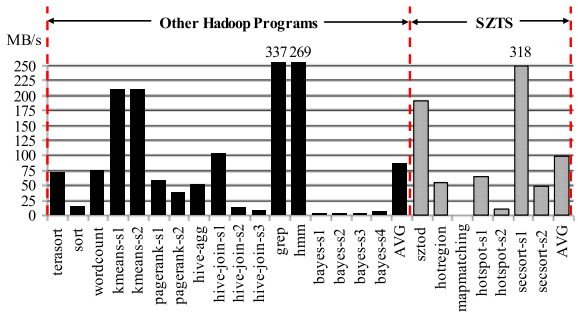


Fig. 10. The data processing speed (DPS) of SZTS and other Hadoop programs (HiBench and CloudRank-D).

see Fig. 9. The same applies to `kmeans-s1` and `kmeans-s2`, and `hmm` and `secsort-s1`. This reconfirms that MKP is indeed an accurate visualization of the workloads' key characteristics.

Second, similar IPC does not necessarily imply similar program behavior. For instance, the IPC values of the `terasort`, `pagerank-s2`, `hmm`, and `bayes-s1` benchmarks are almost the same as shown in Fig. 9. However, the MKPs are significantly different as illustrated in Fig. 11. This indicates that a compound metric such as IPC may conceal underlying behavioral differences, which reinforces the need for a visualization tool such as MKP.

Third, the more important metrics affect IPC and DPS more significantly than less important ones. For example, LLC-MPKI is the most important metric for predicting IPC. Hence, programs with a high IPC are likely to have a low LLC-MPKI: we find that benchmarks `hive-join-s2`, `hive-join-s3`, `bayes-s2`, `bayes-s3`, `bayes-s4`, and `hotspot` have much higher IPC than the other programs, see Fig. 9. We find that these programs indeed have a low LLC-MPKI, see Fig. 11.

5.3.2 MIA Kiviat Plots for DPS

We now analyze job-level behavior in terms of DPS using MKP visualization, see Fig. 12. We make a number of interesting observations here. First, similarly to IPC, similar program behavior observed in the kiviat plots corresponds to similar DPS, see also Fig. 10; but not vice versa. For example, Fig. 12 illustrates that `kmeans-s1` has a similar kiviat plot as `kmeans-s2`, which is confirmed by the same DPS shown in Fig. 10. On the contrary, the DPS of `terasort` is very close to `wordcount`, however their kiviat plots are very different. The same is true for the `hotregion` and `pagerank-s1` benchmarks.

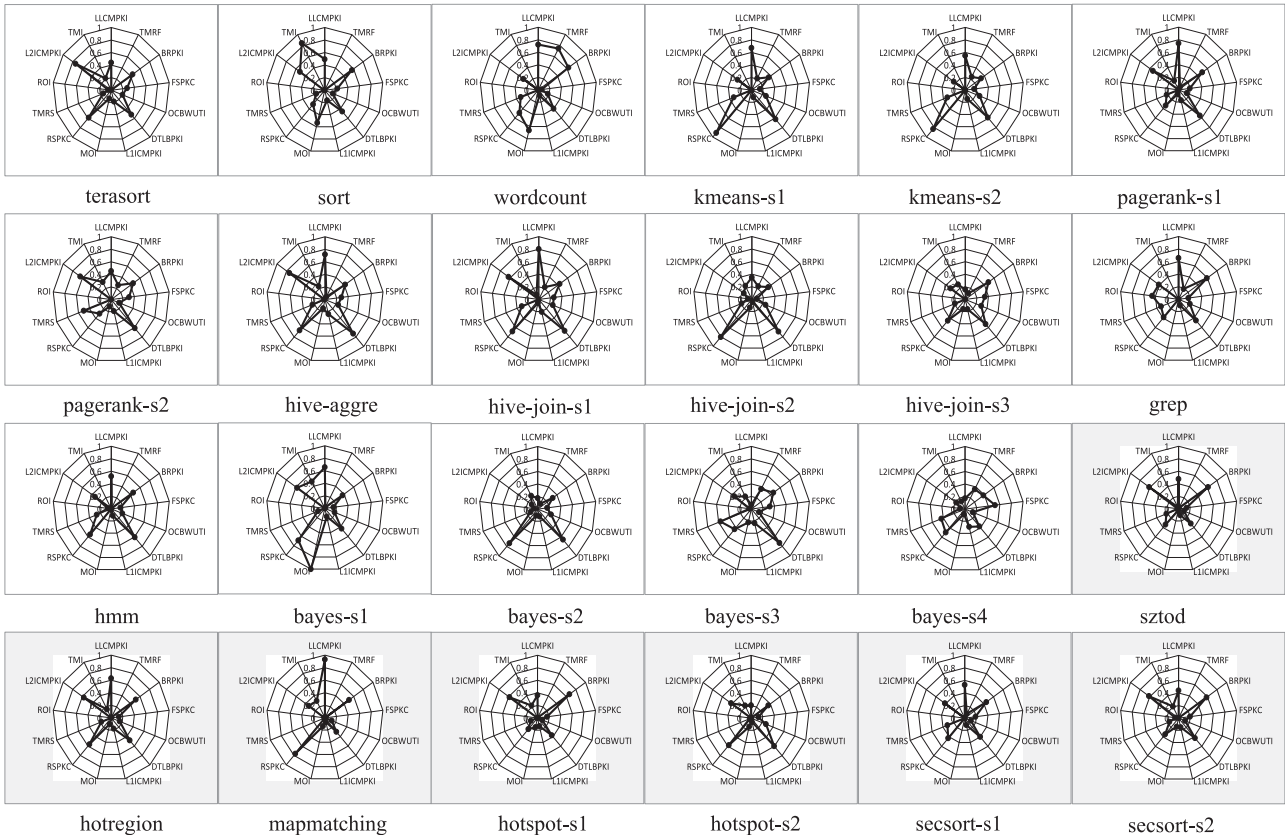


Fig. 11. The MIA kiviat plots for IPC. The most important metric with respect to IPC is shown at the 12 o'clock position, and the importance of metrics decreases following the clockwise direction. The values of the metrics are normalized using Equation (10).

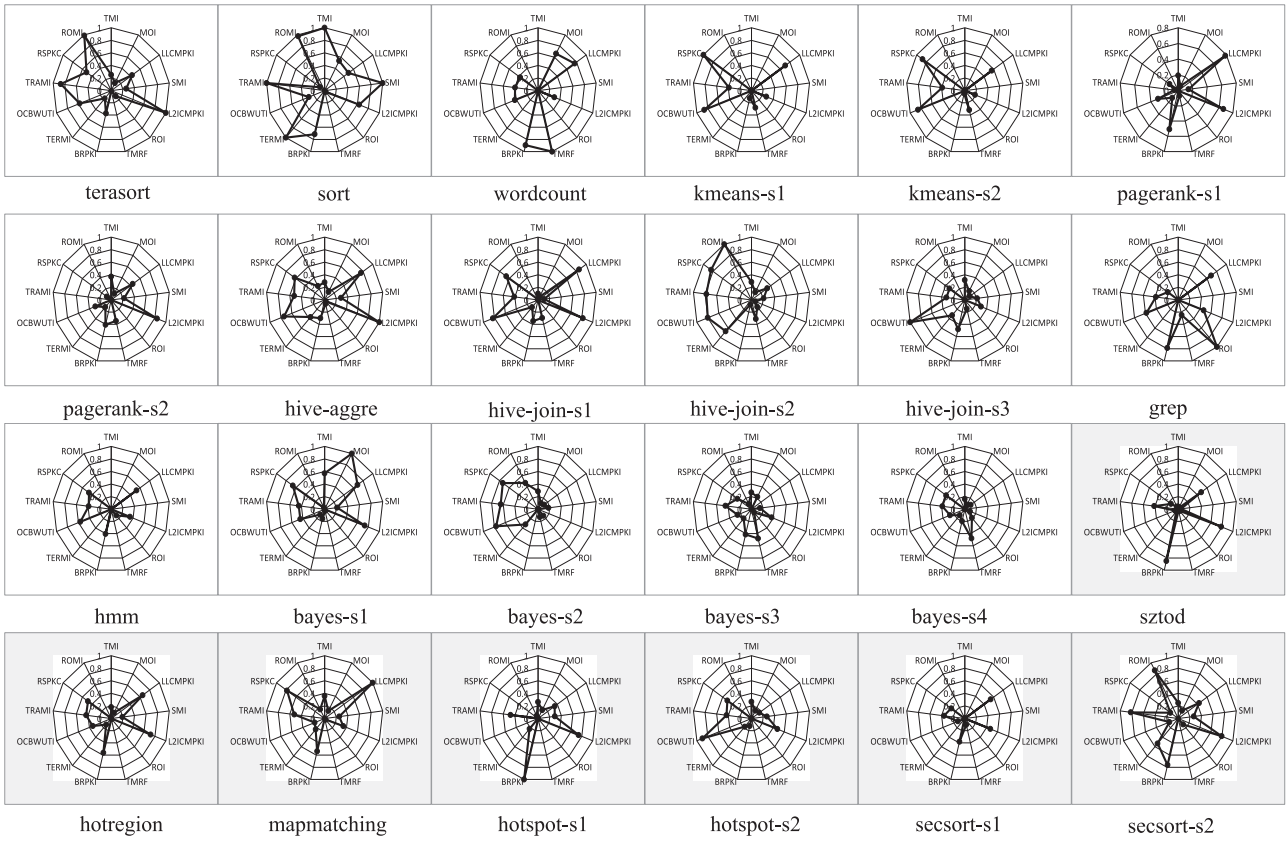


Fig. 12. The MIA kiviat plots for DPS. The most important metric with respect to DPS is shown at the 12 o'clock position, and the importance of metrics decreases following the clockwise direction. The values of the metrics are normalized using Equation (10).

Second, as was the case for IPC as well, the more important metrics in terms of DPS affect DPS more significantly than the less important ones. Fig. 10 reports the highest DPS for `kmeans-s1`, `kmeans-s2`, `grep`, `hmm`, `sztod`, and `secsort-s1`. From MIA, we know that TMI is the most important metric in terms of DPS, and lower is better. We therefore infer that the TMIs of these benchmarks should be low. Looking at Fig. 12, we confirm that TMI is indeed small for these benchmarks. This reconfirms that our MIA identifies the most important metrics.

Third, it is interesting to note that program behavior (dis) similarity in terms of DPS is significantly different from what we observe in terms of IPC, and vice versa. For example, `hotregion`, `secsort-s2`, and `pagerank-s1` exhibit similar behavior (MKP) in terms of IPC while they are significantly different when considering DPS MKP. In contrast, `sztod` and `secsort-s1` are fairly similar to `hmm` in terms of DPS, however `sztod` is quite unique when considering the MKP for IPC. This suggests that only observing program behavior at one layer is not enough to comprehensively understand the characteristics of big data workloads. We must consider and analyze program behavior from multiple layers to obtain a complete performance picture.

5.4 Case Study: SZTS versus Other Hadoop Benchmarks

We now consider a case study to illustrate a potential use case of MIA kiviat plots. We compare the behavior of the SZTS benchmark suite against the other Hadoop benchmarks; the SZTS benchmarks are shown in the gray shaded kiviat plots in Figs. 11 and 12. There are a number of interesting observations to be made here.

First, we find that the area of the MKP for IPC is much smaller for the SZTS benchmarks than for the other Hadoop programs, see Fig. 11. This indicates an interesting insight: the values for several metrics tend to be close to zero. Since most of the metrics in the MKP for IPC are harmful to IPC, we infer that the SZTS benchmarks have a relatively lower IPC than the other Hadoop programs. This is confirmed by the results shown in Fig. 9: only one SZTS benchmark exceeds an IPC of 0.6 while many other Hadoop benchmarks exceeds that—the average IPC for the SZTS benchmarks equals 0.55, in contrast to 0.63 for the other Hadoop programs.

Second, FSPKC (the number of instruction fetch stalls) is close to zero for the SZTS benchmarks, in contrast to some other Hadoop programs, see Fig. 11. This indicates that the SZTS benchmarks exhibit better instruction locality, which results in fewer instruction fetch stalls.

Third, as shown in Fig. 11, most SZTS benchmarks have a relatively high LLC-MPKI compared to the Hadoop programs. Since LLC-MPKI is the most important factor for IPC and lower is better, the IPC of the SZTS benchmarks is lower than that of the other Hadoop programs, which reconfirms the above analysis.

Fourth, the average TMI equals 2.4 for the SZTS benchmarks, whereas that for the other Hadoop programs equals 3.3. This indicates that the average DPS of the SZTS benchmarks is noticeably higher than for the other Hadoop programs because the TMI is the most important factor for DPS and lower is better. Fig. 10 confirms this result.

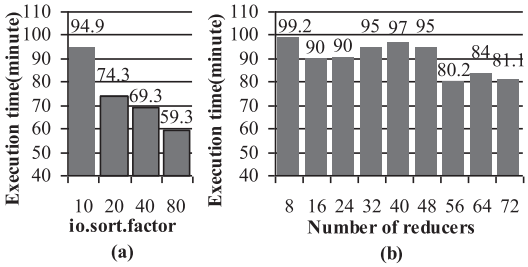


Fig. 13. Execution time optimization for `terasort` by adjusting *io.sort.factor* and *the number of reducers*.

Finally, the conclusion that the area of the MKP of each SZTS benchmark is smaller than those of other Hadoop programs in terms of IPC does not hold true when considering DPS, see Fig. 12. For example, the area of the MKPs for DPS is much larger for `mapmatching` and `secsort-s2` than for `hmm` and `pagerank-s1`.

5.5 Case Study: Efficient Performance Optimization

We now employ another case study to demonstrate the applicability of MIA for steering performance optimization. As shown in Fig. 8, MIA reports that TMI is more important than TMRF with respect to DPS.

By carefully analyzing the relationship between various Hadoop configuration parameters and the TMI and TMRF metrics, we find that the configuration parameters *io.sort.factor* and *the number of reducers* are most tightly related to TMI and TMRF, respectively. *io.sort.factor* specifies how many fragmented files on disk can be merged at once. A small value of it leads to a large amount of TMI because some of the fragmented files are re-read and re-written multiple times when the files are merged. *the number of reducers* directly affects the execution time of reduce functions, deciding the value of TMRF.

Taking the `terasort` benchmark with 400 GB of input data as an example, we adjust the values of *io.sort.factor* and *the number of reducers* to observe its execution time variation. As shown in Fig. 13, execution time decreases from 94.9 minutes to 59.3 minutes when increasing the value for *io.sort.factor* from 10 (default) to 80, reflecting a 37.5 percent reduction in execution time. In contrast, the execution time decreases from 99.2 to 80.2 minutes only when increasing the *number of reducers* from 8 to 72, implying an execution time reduction of merely 19.2 percent. Without MIA, one may take a lot of effort tuning the *number of reducers* instead of tuning *io.sort.factor*, achieving less performance improvement. This indicates that MIA can help users optimize workloads more efficiently by focusing on the more important metrics.

Note that it only takes us half of an hour effort to identify the tight relationship between *io.sort.factor* and TMI by reading the Apach Hadoop reference guide about configurations [53]. The guide describes that *io.sort.factor* specifies the number of fragmented files that can be merged at once. If its value is small, it causes a large number of merge operations and in turn produces a large amount of accumulated temporary data (TMI). Therefore, we infer that *io.sort.factor* is important because we know TMI is important with the help of MIA. Moreover, the relationship between the configuration parameters and the job metric is static and does not vary across benchmarks, which can be incorporated with MIA to efficiently optimize Hadoop programs.

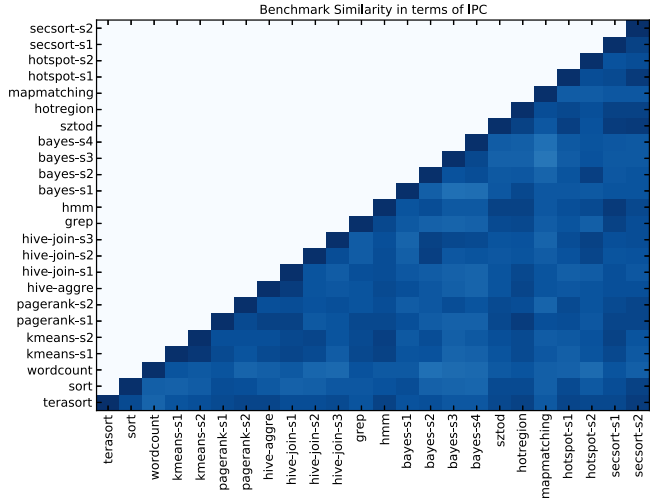


Fig. 14. Benchmark similarity matrix for IPC. A dark point indicates that the two benchmarks (represented by the corresponding X and Y points) are similar in terms of IPC.

5.6 Benchmark Similarity Matrices

We now summarize the (dis)similarity of all the experimented benchmarks using Benchmark Similarity Matrix visualization. Fig. 14 shows the BSM in terms of IPC. BSM visualization provides an intuitive perspective for analyzing the (dis)similarity among workloads. For example, the `terasort` benchmark is quite different from all other benchmarks (light blue boxes at the bottom row), while being quite similar to `secsort-s2`, see the rightmost point at the bottom row. The `sort` benchmark (row above `terasort`) is even more different from the other programs but is slightly less similar to `secsort-s2` compared to `terasort`. The `wordcount` benchmark is the outlier benchmark here: its behavior is significantly different from the other benchmarks because there are no dark points in the third row and third column. Also `mapmatching` is significantly different from the other benchmarks.

Fig. 15 provides a similar view for DPS. The `sort` benchmark is the outlier at the system level because the respective row and column are the lightest. The second most outlier benchmark is `bayes-s1`. Interestingly, these two benchmarks did not appear as the most extreme outliers at the node level (see the BSM for IPC). This reconfirms that characterizing big data workloads in a comprehensive way requires analyzing performance at multiple layers in the system stack.

5.7 Generalization of MIA

We now discuss the generalization of MIA. We build the performance model using an ensemble learning algorithm, Stochastic Gradient Regression Tree, based on the training set produced by 15 programs, each with more than 5 different input datasets. Therefore, the identified important metrics are the same for all the experimented program-input pairs. When we study a new program, if the characteristics of the program are similar to those of one of the experimented program (using the technique described in Section 3.5 to measure the similarity between two programs), it is

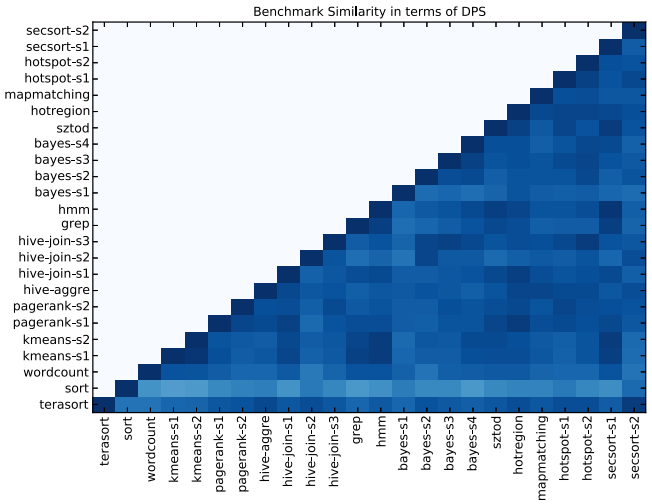


Fig. 15. Benchmark similarity matrix for DPS. A dark point indicates that the two benchmarks (represented by the corresponding X and Y points) are similar with respect to DPS.

unnecessary to retrain the model. The important metrics can also be applied to the new program. However, if the new program is significantly different from the experimented ones which produce the training set, we need to retrain the model by adding the vectors defined by Equations (2) and (3) to matrices (4) and (5), respectively. In such a case, the model retraining is inevitable for all machine learning and the like modeling techniques.

Moreover, MIA can also be used to build an accurate performance model and identify the important metrics for a single program as long as a large enough training set can be collected. This can be done by running the program with a large enough number of different input datasets and each run collects the vectors defined by Equations (2) and (3), respectively.

6 Related Work

6.1 Workload Characterization

Although workload characterization is extremely important for computing system design and performance evaluation, there are only few of studies focusing on the methodology itself. Eeckhout et al. [17] propose a Principal Component Analysis (PCA) based approach to characterize workloads for computer architecture research. While PCA is a good way to identify the main characterization of a workload, it obfuscates information from the original metrics because the principal components are computed as linear combinations of the original metrics. In contrast, MIA identifies the most important metrics among the original metrics, which yields a more intuitive and comprehensive way to analyze and understand workload behavior.

Hoste and Eeckhout [18] propose a linkage-clustering based dendrogram to visualize the (dis)similarity among workloads. The difference in workload behavior is computed using the principal components. One limitation of this repesentation is that it does not illustrate why two workloads are dissimilar from each other. Indeed, a large linkage distance implies dissimilar behavior, however, there could be various reasons that are not apparent from the linkage distance. The MIA-based Kiviat Plots, on the other hand, provide a comprehensive picture explaining why two workloads are (dis)similar.

A number of previous works use Kiviat plots to visualize workload characterization results. For example, Zhang et al. [48] use Kiviat plots to show the envelop of eight metrics for many-task computing programs. Che et al. [49] employ Kiviat plots to compare the (dis)similarity of different GPGPU workloads. None of these studies however explain how and why they choose the specific metrics they choose; MIA on the other hand selects the most important workload metrics in a systematic way.

6.2 Big Data Workload Analysis

Big data processing places unprecedented demands on computing systems, which makes evaluating and understanding big data systems challenging. Benchmarking big data systems therefore attracts significant attention.

The traditional benchmark suite for database systems is TPC [5]. Big data imposes severe challenges on database systems and impels these systems to improve for big data processing, see TPC-DS [5] and BigBench [11]. Another database benchmark suite for big data processing is YCSB which is designed for evaluating Yahoo!'s cloud platform and data storage systems [7]. YCSB mainly consists of online service workloads—so-called 'Cloud OLTP' workloads. More recently, Armstrong et al. [8] released the LinkBench benchmark suite based on Facebook's social graph. These benchmark suites focus on database systems which are fairly different from Hadoop-based systems.

Several big data benchmark suites have been proposed that do not focus on databases exclusively. Ferdman et al. [9] study the microarchitecture-level characteristics of scale-out workloads in the cloud. Wang et al. [10] propose a big data benchmark suite named BigDataBench for evaluating Internet services. Jia et al. [12] characterize data analysis workloads in the data center. Early Hadoop benchmarks include `GridMix` [13], `Sort` [14], and `TeraSort` [15], [16]. Zhang et al. [26] propose a parameterizable benchmarking framework for MapReduce programs [26]. The closest related works are HiBench [1], CloudRank-D [2], and SZTS [25]. None of these prior works do provide a way to identify the important workload characteristics. In this work, we propose MIA to do so.

7 Conclusions

Big data workloads and systems pose significant benchmarking challenges as it requires performance metrics to be measured at various levels in the system stack. These performance metrics are correlated, further complicating the analysis. In this paper, we propose a novel ensemble-learning based methodology to quantify the importance of metrics, named Metric Importance Analysis. We leverage MIA and propose MIA-based Kiviat plots and the Benchmark Similarity Matrix to visualize a workload's characteristics and ease the comparison of the behavior among a set of workloads. Our workload characterization shows that MIA indeed is a powerful tool that reveals insight, and provides intuitive and visually easy-to-grasp information about the program behavior (dis)similarity among big data workloads.

ACKNOWLEDGMENTS

We thank the reviewers for their thoughtful comments and suggestions. This work is supported by the National Key R&D Program of China under no. 2016YFB1000204; NSFC under grants no. 61672511, 61702495; outstanding technical talent program of CAS. Additional support is provided by the major scientific and technological project of Guangdong province (2014B010115003), Shenzhen Technology Research Project (JSGG20160510154–636747), and Key technique research on Haiyun Data System of NICT, CAS under grant no. XDA06010500. Lieven Eeckhout is partly supported by a Chinese Academy of Sciences (CAS) visiting professorship for senior international scientists.

REFERENCES

[1] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The HiBench benchmark suite: Characterization of the MapReduce-based data analysis," in *Proc. IEEE 26th Int. Conf. Data Eng. Workshops*, 2010, pp. 41–51.

[2] C. Luo, et al., "Cloudrank-D: Benchmarking and ranking cloud computing systems for data processing applications," *Frontiers Comput. Sci.*, vol. 6, no. 4, pp. 347–362, 2012.

[3] Y. Lou, C. Zhang, Y. Zheng, X. Xie, W. Wang, and Y. Huang, "Map-matching for low-sampling-rate GPS trajectories," in *Proc. 17th ACM SIGSPATIAL Int. Conf. Advances Geographic Inf. Syst.*, 2009, pp. 352–361.

[4] C. Bienia, S. Kumar, J. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architecture implications," in *Proc. Int. Conf. Parallel Archit. Compilation Techn.*, 2008, pp. 72–81.

[5] Active TPC Benchmarks. [Online]. Available: http://www.tpc.org/information/ benchmarks.asp, Accessed on: May 12, 2017.

[6] Apach Hive. [Online]. Available: http://hive.apache.org/, Accessed on: May 12, 2017.

[7] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proc. 1st ACM Symp. Cloud Comput.*, 2010, pp. 143–154.

[8] T. G. Armstrong, V. Ponnekanti, D. Borthakur, and M. Callaghan, "LinkBench: A database benchmark based on the Facebook social graph," in *Proc. SIGMOD Conf.*, 2013, pp. 1185–1196.

[9] M. Ferdman, et al., "Clearing the clouds: A study of emerging workloads on modern hardware," in *Proc. Annu. Int. Conf. Archit. Support Program. Languages Operating Syst.*, 2012, pp. 37–48.

[10] L. Wang, et al., "BigDataBench: A big data benchmark suite from internet services," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit.*, 2014, pp. 488–499.

[11] A. Ghazal, et al., "BigBench: Towards an industry standard benchmark for big data analytics," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2013, pp. 1197–1208.

[12] Z. Jia, L. Wang, J. Zhan, L. Zhang, and C. Luo, "Characterizing data analysis workloads in data centers," in *Proc. IEEE Int. Symp. Workload Characterization*, 2013, pp. 66–76.

[13] GridMix Program. Available in Hadoop source distribution, [Online]. Available: src/benchmarks/gridmix

[14] Sort program. Available in Hadoop source distribution, [Online]. Available: src/examples/org/apache/hadoop/examples/sort

[15] TeraSort. [Online]. Available: http://sortbenchmark.org/, Accessed on: May 12, 2017.

[16] Hadoop TeraSort program. Available in Hadoop source distribution since 0.19 version, [Online]. Available: src/examples/org/apache/hadoop/examples/terasort

[17] L. Eeckhout, H. Vandierendonck, and K. De Bosschere, "Quantifying the impact of input data sets on program behavior and its applications," *J. Instruction-Level Parallelism*, vol. 5, no. 1, pp. 1–33, 2003.

[18] K. Hoste and L. Eeckhout, "Microarchitecture-independent workload characterization," *IEEE Micro*, vol. 27, no. 3, pp. 63–72, May/Jun. 2007.

[19] C. Baru, M. Bhandarkar, R. Nambiar, M. Poess, and T. Rabl, "Benchmarking big data systems and the BIGDATA TOP100 list," *Big Data J.*, vol. 1, no. 1, pp. 60–64, 2013.

[20] The Fifth Workshop on Big Data Benchmarking. (2014). [Online]. Available: http://clds.sdsc.edu/wbdb2014.de, Accessed on: May 12, 2017.

[21] BPOE-5: The Fifth Workshop on Big Data Benchmarks, Performance, and Emerging Hardware. (2014). [Online]. Available: http://prof.ict.ac.cn/bpoe_5_vldb, Accessed on: May 12, 2017.

[22] C. Baru, M. Bhandarkar, R. Nambiar, M. Poess, and T. Rabl, "Setting the direction for big data benchmark standards," *Sel. Topics Perform. Eval. Benchmarking*, vol. 7755, pp. 197–208, 2013.

[23] J. H. Friedman, "Stochastic gradient boosting," *Comput. Statist. Data Anal.*, vol. 38, no. 4, pp. 367–378, 2002.

[24] J. H. Friedman and J. J. Meulman, "Multiple additive regression trees with application in epidemiology," *Statist. Med.*, vol. 22, no. 9, pp. 1365–1381, 2003.

[25] W. Xiong, et al., "A characterization of big data benchmarks," in *Proc. IEEE Int. Conf. Big Data*, 2013, pp. 118–125.

[26] Z. Zhang, L. Cherkasova, and B. T. Loo, "Parameterizable benchmarking framework for designing a MapReduce performance model," *J. Concurrency Comput.: Practice Experience*, vol. 26, pp. 2005–2026, 2014.

[27] W. Xiong, Z. Yu, L. Eeckhout, Z. Bei, F. Zhang, and C. Xu, "SZTS: A novel big data transportation system benchmark suite," in *Proc. 44th Int. Conf. Parallel Process.*, 2015, pp. 819–828.

[28] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. 6th Symp. Operating Syst. Des. Implementation*, 2004, pp. 137–150.

[29] Apache Hadoop. [Online]. Available: http://hadoop.apache.org/, Accessed on: May 12, 2017.

[30] J. M. Calcagni, "Shape in ranking Kiviat graphs," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 5, no. 1, pp. 35–37, Jan. 1976.

[31] M. F. Morris, "Kiviat graphs: Conventions and figures of metric," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 3, no. 3, pp. 2–8, Oct. 1974.

[32] H. E. Barry Merrill , "A technique for comparative analysis of Kiviat graphs," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 3, no. 1, pp. 34–39, Mar. 1974.

[33] H. W. Barry Merrill , "Further comments on comparative evaluation of Kiviat graphs," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 4, no. 1, pp. 1–10, Jan. 1975.

[34] K. W. Kolence and P. J. Kiviat, "Software unit profiles & Kiviat figures," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 2, no. 3, pp. 2–12, Sep. 1973.

[35] D. A. Freedman, *Statistical Models: Theory and Practice*. Cambridge, U.K.: Cambridge Univ. Press. 2009.

[36] N. John and W. Robert, "Generalized linear models," *J. Roy. Statist. Soc.*, vol. 135, no. 3, pp. 370–384, 1972.

[37] I. J. Hastie and R. J. Tibshirani, *Generalized Additive Models*. London, U.K.: Chapman & Hall/Boca Raton, FL: CRC. 1990.

[38] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–295, 1995.

[39] M. Minsky and S. Papert, *An Introduction to Computational Geometry*. Cambridge, MA, USA: MIT Press, 1969.

[40] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Ann. Statist.*, vol. 29, pp. 1189–1232, 2001.

[41] J. A. Poovey, T. M. Conte, M. Levy, and S. Gal-On, "A benchmark characterization of the EEMBC benchmark suite," *IEEE Micro*, vol. 29, no. 5, pp. 18–29, Sep./Oct. 2009.

[42] W. Jia, K. A. Shaw, and M. Martonosi, "Stargazer: Automated regression-based GPU design space exploration," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, 2012, pp. 2–13.

[43] W. Jia, K. A. Shaw, and M. Martonosi, "Starchart: Hardware and software optimization using recursive partitioning regression trees," in *Proc. 22nd Int. Conf. Parallel Archit. Compilation Techn.*, 2013, pp. 257–267.

[44] W. Jia, K. A. Shaw, and M. Martonosi, "GPU performance and power tuning using regression trees," *ACM Trans. Archit. Code Optimization*, vol. 12, no. 2, 2015, Art. no. 13.

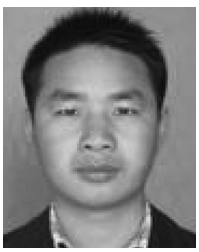
[45] A. E. Gencer, D. Bindel, E. G. Sirer, and R. V. Renesse, "Configuring distributed computations using response surfaces," in *Proc. Annu. ACM/IFIP/USENIX Middleware Conf.*, 2015, pp. 235–246.

[46] E. Ipek, S. A. McKee, R. Caruana, B. R. de Supinski, and M. Schulz, "Efficiently exploring architectural design spaces via predictive modeling," in *Proc. 12th ACM Int. Conf. Archit. Support Program. Languages Operating Syst.*, 2006, pp. 195–206.

[47] A. Spark Team, Aparch Spark. 2016. [Online]. Available: http://spark.apache.org/

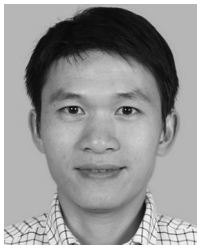
[48] Z. Zhang, D. S. Katz, M. Wilde, J. M . Wozniak, and I. Foster, "MTC envelope: Defining the capability of large scale computers in the context of parallel scripting applications," in *Proc. ACM Int. Symp. High-Perform. Parallel Distrib. Comput.*, 2013, pp. 37–48.
[49] S. Che, et al., "Rodinia: A benchmark suite for heterogeneous computing," in *Proc. IEEE Int. Symp. Workload Characterization*, 2009, pp. 44–54.
[50] F. Azmandian, M. Moffie, J. G. Dy, J. A. Aslam, and D. R. Kaeli, "Workload characterization at the virtualization layer," in *Proc. 19th Annu. Int. Symp. Model. Anal. Simul. Comput. Telecommun. Syst.*, 2011, pp. 63–72.
[51] H. H. Huang, S. Li, A. Szalay, and A. Terzis, "Performance modeling and analysis of flash-based storage devices," in *Proc. 27th IEEE Symp. Mass Storage Syst. Technol.*, 2011, pp. 1–11.
[52] L. Palden and Z. Xiaobo, "AROMA: Automated resource allocation and configuration of MapReduce environment in the cloud," in *Proc. 9th ACM Int. Conf. Autonomic Comput.*, 2012, pp. 63–72.
[53] [Online]. Available: https://hadoop.apache.org/docs/r1.0.4/mapred-default.html

Zhibin Yu received the PhD degree in computer science from Huazhong University of Science and Technology (HUST), in 2008. Now he is a professor in the SIAT, CAS. His research interests include computer architecture, workload characterization and generation, performance evaluation, multi-core architecture, GPGPU architecture, big data processing and so forth. He won the outstanding technical talent program of Chinese Academy of Science (CAS), in 2014. He also won the first award in teaching contest of HUST young lectures in 2005 and the second award in teaching quality assessment of HUST in 2003. He serves for ISCA, MICRO, and HPCA.



Wen Xiong received the BS degree from Wuhan Institute of Technology, in 2005 and the MS degree from HuaZhong University of Science and Technology, in 2008. He is working toward the PhD degree in Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences. His research focuses on big data benchmarking and distributed storage system.

Lieven Eeckhout received the PhD degree from Ghent University, in 2002. He is a professor with the Ghent University, Belgium. His research interests include computer architecture with a specific emphasis on performance evaluation methodologies and dynamic resource management. He is the recipient of the 2017 ACM SIGARCH Maurice Wilkes Award. His work has been awarded with two IEEE Micro Top Pick awards and a Best Paper Award at ISPASS 2013. He published a Morgan & Claypool synthesis lecture monograph in 2010 on performance evaluation methods. He was the program chair for HPCA 2015, CGO 2013 and ISPASS 2009; and currently serves as the editor-in-chief of the *IEEE Micro* and as associate editor of the *ACM Transactions on Architecture and Code Optimization* and the *IEEE Transactions on Computers*.



Zhendong Bei received the BS degree from National University of Defense Technology, in 2006 and the MS degree from Central South University, in 2009. He is working toward the PhD degree in Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences. His research interests include performance optimization of big data system, data mining, machine learning, and image processing.

Avi Mendelson received the BSc and MSc degrees from the CS Department, Technion, in 1979 and 1982, respectively, and the PhD degree from the University of Massachusetts at Amherst (UMASS), in 1990. He is a professor in the CS and EE Departments Technion, and the head of the EE Department at the School of Engineering, Kinneret College, Israel. He has a blend of industrial and academic experience. As part of his industrial role, he worked for Intel 11 years, where he served as a senior researcher and principle engineer in the Mobile Computer Architecture Group, in Haifa Israel. While in Intel, he was the chief architect of the CMP (multi-core-on-chip) feature of the first dual core processors Intel developed. His research interests span over different areas such as computer architecture, operating systems, power management, reliability, fault-tolerance, cloud computing, HPC, and GPGPU.

Chengzhong Xu received the PhD degree from the University of Hong Kong, in 1993. He is currently a tenured professor of Wayne State University and the director of the Institute of Advanced Computing and Data Engineering of Shenzhen Institute of Advanced Technology of Chinese Academy of Sciences. His research interests include parallel and distributed systems and cloud computing. He has published more than 200 papers in journals and conferences. He serves on a number of journal editorial boards, including the *IEEE Transactions on Computers*, the *IEEE Transactions on Parallel and Distributed Systems*, the *IEEE Transactions on Cloud Computing*, the *Journal of Parallel and Distributed Computing* and the *China Science Information Sciences*.