

Characterizing Data Analysis Workloads in Data Centers

Zhen Jia^{1,2}, Lei Wang^{1,2}, Jianfeng Zhan^{1*}, Lixin Zhang¹, and Chunjie Luo¹

¹State Key Laboratory Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences

²University of Chinese Academy of Sciences, China

¹{jiazhen, wl}@ncic.ac.cn, {zhanjianfeng, zhanglixin, luochunjie}@ict.ac.cn

Abstract—As the amount of data explodes rapidly, more and more corporations are using data centers to make effective decisions and gain a competitive edge. Data analysis applications play a significant role in data centers, and hence it has become increasingly important to understand their behaviors in order to further improve the performance of data center computer systems.

In this paper, after investigating three most important application domains in terms of page views and daily visitors, we choose eleven representative data analysis workloads and characterize their micro-architectural characteristics by using hardware performance counters, in order to understand the impacts and implications of data analysis workloads on the systems equipped with modern superscalar out-of-order processors. Our study on the workloads reveals that data analysis applications share many inherent characteristics, which place them in a different class from desktop (SPEC CPU2006), HPC (HPCC), and service workloads, including traditional server workloads (SPECweb2005) and scale-out service workloads (four among six benchmarks in CloudSuite), and accordingly we give several recommendations for architecture and system optimizations.

On the basis of our workload characterization work, we released a benchmark suite named *DCBench* for typical datacenter workloads, including data analysis and service workloads, with an open-source license on our project home page on <http://prof.ict.ac.cn/DCBench>. We hope that *DCBench* is helpful for performing architecture and small-to-medium scale system researches for datacenter computing.

Keywords—Datacenter workloads; Workload characterization; Benchmarking

I. INTRODUCTION

In the context of digitalized information explosion, more and more businesses are analyzing massive amount of data – so-called big data – with the goal of converting big data to “big value”. Data center workloads can be classified into two categories: services and data analysis workloads as mentioned in [1] and [2]. Typical data analysis workloads include business intelligence, machine learning, bio-informatics, and ad hoc analysis [3][4].

The business potential of the data analysis applications is a driving force behind the design of innovative data center systems including both hardware and software[5] [6] [7]. For example, the recommendation system is a typical example

with huge financial implications, aiming at recommending the right products to the right buyers by mining user behavior and other logs. For data analysis in data centers is a relatively new but very important application area, there is a need to understand various representative workloads’ performance characteristics and what optimizations will further improve the performance. So characterizing data analysis workloads is meaningful for system designers and researchers to gain insights on optimizing data center computer systems.

There have been much work proposed to evaluate data mining algorithms or so-called scale-out cloud workloads in different aspects, such as MineBench [8], HiBench [9], and CloudSuite [10]. The state-of-the-art work of characterizing data center workloads on a micro architecture level is CloudSuite [10]. However, CloudSuite is biased towards online service workloads: among six benchmarks, there are four scale-out service workloads, (including *Data Serving*, *Media Streaming*, *Web Search*, *Web Serving*), and only one data analysis workload—*Naive Bayes*. Our work show the data analysis workloads are significantly diverse in terms of both speedup performance (Section II-B) and micro-architectural characteristics (Section IV). In a word, only one application is not enough to represent various categories of data analysis workloads.

In this paper, firstly we single out three important applications domains in Internet services: *search engine*, *social networks*, and *electronic commerce* (listed in Figure 1) according to a widely acceptable metrics — the number of page views and daily visitors. And then, we choose eleven representative data analysis workloads (especially intersection workloads) among the three applications domains. In comparison with that of CloudSuite described in [10], our experiment approach is more pragmatic. First, we deploy a larger input data set varying from 147 to 187 GB that are stored in both the memory and disk systems instead of completely storing data (only 4.5GB for *Naive Bayes* in [10]) in the memory system. Second, for each workload, we collect the performance data of the whole run time after the warm-up instead of a short period (180 seconds in [10]).

Our study reveals that data analysis applications share many inherent characteristics, which place them in a different class from desktop (SPEC CPU2006), HPC (HPCC),

* The corresponding author is Jianfeng Zhan.

and traditional server workload (SPECweb2005), and scale-out service workloads (four among six benchmarks in CloudSuite). Meanwhile, we also observe that the scale-out service workloads (four among six benchmarks in CloudSuite) share many similarity in terms of micro-architectural characteristics with that of SPECweb2005, so in the rest of this paper, we just use the service workloads to describe them. On the basis of our workload characterization work, we released a benchmark suite named *DCBench* for datacenter computing. Our key findings are as follows:

First, the data analysis workloads have higher IPC than that of the services workloads while lower than that of computation-intensive HPC workloads, e.g., *HPC-HPL*, *HPC-DGEMM*.

Second, corroborating previous work [10], both the data analysis workloads and service workloads suffer from notable front-end stalls, which may be caused by two factors: deep memory hierarchy with long latency in modern processor [10], and large binary size complicated by high-level language and third-party libraries. However, we note the significant differences between the data analysis workloads and the service workloads in terms of stall breakdown: the data analysis workloads suffer more stalls in the out-of-order part of the pipeline (about 57% on average), while the service workloads suffers more stalls before instructions entering the out-of-order part (about 73% on average).

Third, the third-party libraries and high-level languages used by the data center workloads may aggravate the inefficiency of L1 instruction cache and instruction TLB. So when writing the program (with the support of third-party libraries and high-level languages), the engineers should pay more attention to the code size.

Fourth, for the data analysis workloads, L2 cache is acceptably effective, and they have lower L2 cache misses (about 11 L2 cache misses per thousand instructions on average) than that of the service workloads (about 60 L2 cache misses per thousand instructions on average) while higher than that of the HPC workloads. Meanwhile, for the data analysis and service workloads, on the average 85.5% and 94.9% of L2 cache misses are hit in L3 cache, respectively. Considering modern processors dedicate approximately half of the die area to caches, optimizing the LLC capacity will improve the energy-efficiency of processor and save the die size. For the service workloads, our observation corroborate the previous work [10]: L2 cache is ineffective.

Finally, for the data analysis workloads, the misprediction ratios are lower than that of most of the service workloads, which implies that the branch predictor of modern processor is good. A simpler branch predictor may be preferred so as to save power and die area.

The remainder of the paper is organized as follows. Section II describes our benchmarking methodology and design decisions. Section III states our experiment methodology. Section IV presents the micro-architectural characteristics

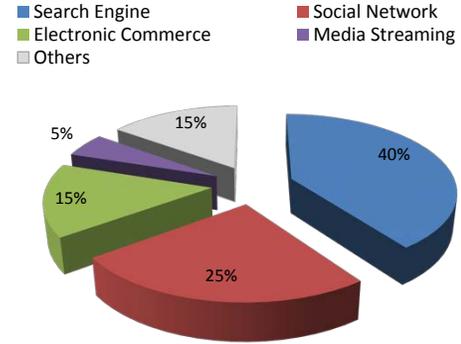


Figure 1. Top sites in the web [11].

of the data analysis workloads in comparison with other benchmarks suites. Section V introduces our *DCBench*. Section VI draws conclusions of the full paper and mentions the future work.

II. BENCHMARKING METHODOLOGY AND DECISIONS

A. Workloads Requirements

The workloads we choose must meet the following requirements.

Representative

There are many representative workloads in many specific fields, such as SPEC CPU for processors, SPECweb for Web servers. Those workloads are representative in their own fields. For data analysis field in data center, we should choose representative workloads in the most important application domains.

Distributed

In general, most of data analysis workloads in data center are distributed on several nodes for the large amount of data can not be processed efficiently in a single node. An application running on a single computer can not represent the applications in real world data centers.

Employ State-of-art Techniques [12]

In data centers, workloads change frequently, which Barroso et al. call workload churns [2]. So the workloads we want to characterize should be recently used and implemented with emerging techniques in different domains.

B. Can one data analysis workload represent all?

In the rest of this paper, in comparison with other benchmarks, we find that data analysis workloads and scale-out service workloads have some common characteristics, e.g, notable frond-end stalls, which is consist with CloudSuite. However, CloudSuite is biased towards service workloads, and they choose four service workloads among the six benchmarks and only include one data analysis workload: *Naive Bayes*. Can *Naive Bayes* workload represent all of the data analysis workloads? The answer is no.

In this subsection, we perform the experiments on a 9-node Hadoop cluster, including one master node. The

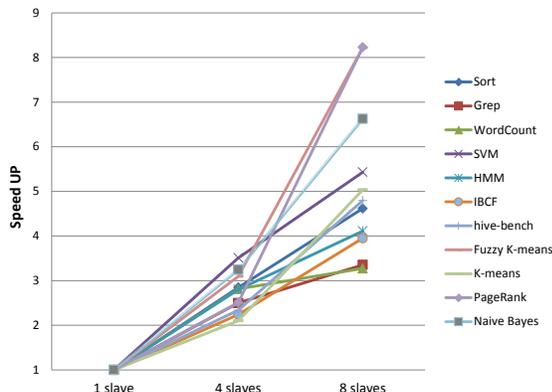


Figure 2. Varied speed up performance of eleven data analysis workloads. Only Bayes is included into CloudSuite.

configuration of the system and the Hadoop environment is the same as that in Section III. For the conciseness, we leave the configuration details in Section III. We choose eleven representative workloads including *Naive Bayes*, the details of which can be found at Section II-C, and use the same input data sets (listed in Table I). We change the number of slave nodes from 1 to 8, using the run time on the one-slave system as a baseline performance data, and then calculate the speed up. As shown in Figure. 2, for different workloads, the speed up data on an eight-node system range from 3.3 to 8.2, and the value of *Naive Bayes* is 6.6, which indicates that the data analysis workloads are diverse in terms of performance characteristics. Meanwhile our experiments in Section IV also show that different data analysis applications have different micro-architecture level characteristics.

In a word, only one application is not enough to represent various categories of data analysis workloads.

C. Workloads Choosing

In order to find workloads which meet all the requirements listed in Section II-A, we firstly decide and rank the main application domains according to a widely acceptable metric—the number of pageviews and daily visitors, and then single out the main applications from the most important application domains. We investigate the top sites listed in Alexa [11], of which the rank of sites is calculated using a combination of average daily visitors and page views. We classified the top 20 sites into 5 categories including search engine, social network, electronic commerce, media streaming and others. Figure 1 shows the categories and their respective share. To keep concise and simple, we focus on the top three application domains: *search engine*, *social networks* and *electronic commerce*.

We choose the most popular applications in those three application domains. Table II shows application scenarios of each workload, which is characterized in this paper, indicating most of our chosen workloads are intersections

among three domains.

1) *Basic operations*: *Sort*, *WordCount* and *Grep* are three basic operations which are frequently used in data analysis fields. Several researchers also use them in their work [13]. *Sort* ranks the records by their key. *WordCount* reads text files and counts the number of occurrences of each word. *Grep* extracts matching strings from text files and counts the number of the occurrence of the matching strings.

2) *Classification and Clustering*: The classification and clustering algorithms are also widely used in those three application domains. The goods information in electronic commerce and key words of search engines are the input data of classification and clustering applications. We choose *Naive Bayes* and *Support Vector Machine* (in short, *SVM*) from numerous classification algorithms. *Naive Bayes* is a simple probabilistic classifier which applies Bayes' theorem with strong (naive) independence assumptions. *SVM*, a very widely used classification algorithm, maps the training samples as points in a high dimension space so that the samples of the separate categories can be divided by a clear gap. And new samples are categorized based on the side of the gap they fall on.

For the clustering applications, we choose *K-means* and *Fuzzy K-means* algorithms because those two clustering algorithms are so famous in many application scenarios. *K-means* partitions observations into a fixed number clusters in which each observation belongs to the (only) cluster with the nearest mean. *Fuzzy K-means* is an extension of *K-means*, but quite different from the latter. It is statistically formalized and allows soft clusters where an observation can belong to multiple clusters with given probabilities.

3) *Recommendation*: The recommendation algorithms are widely used to recommend goods, friends, and key words in electronic commerce, social network and search engine, respectively. We choose an *Item Based Collaborative Filtering* (in short *IBCF*) application. It estimates a user's preference towards an item by looking at his/her preferences towards related items.

4) *Segmentation*: Segmentation is very important for web search, especially for a language like Chinese. We implement a segmentation algorithm using the *Hidden Markov Model* (in short, *HMM*). *HMM* is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved hidden states.

5) *Graph calculation*: *PageRank* [14] is a typical graph calculation algorithm on link analysis, and *PageRank-like* algorithms are frequently used in search engine.

6) *Data warehouse operations*: In the three application domains we mentioned above, data warehouse are often used to manage data. The previously published *Hive-bench* [15] is a benchmark for the data warehouse operations based on the *Hive* [16]. Here, we include a series of representative SQL-like statements in *Hive-bench* as a part of our workloads.

Table I
REPRESENTATIVE DATA ANALYSIS WORKLOADS

No.	Workload	Input Data Size	#Retired Instructions (Billions)	Source
1	Sort	150 GB documents	4578	Hadoop example
2	WordCount	154 GB documents	3533	Hadoop example
3	Grep	154 GB documents	1499	Hadoop example
4	Naive Bayes	147 GB text	68131	mahout[17]
5	SVM	148 GB html file	2051	our implementation
6	K-means	150 GB vector	3227	mahout
7	Fuzzy K-means	150 GB vector	15470	mahout
8	IBCF	147 GB ratings data	32340	mahout
9	HMM	147 GB html file	1841	our implementation
10	PageRank	187 GB web page	18470	mahout
11	Hive-bench	156 GB DBtable	3659	Hivebench

Table II
SCENARIOS OF DATA ANALYSIS.

Name	Domain	Scenarios
Grep	search engine	Log analysis
	social network	Web information extraction
	electronic commerce	Fuzzy search
Bayes	social network	Spam recognition
	electronic commerce	Web page classification
SVM	social network	Image Processing
	electronic commerce	Data Mining
		Text Categorization
PageRank	search engine	Compute the page rank
Fuzzy K-means	search engine	Image processing
	social network	High-resolution landform classification
K-means	electronic commerce	classification
HMM	social network	Speech recognition
	search engine	Word Segmentation
		Handwriting recognition
WordCount	search engine	Word frequency count
	social network	Calculating the TF-IDF value
	electronic commerce	Obtaining the user operations count
Sort	electronic commerce	Document sorting
	search engine	Pages sorting
	social network	

III. EXPERIMENTAL SETUP

This section firstly describes the experimental environments on which we conduct our study, and then explains our experiment methodology.

A. Hardware Configurations

We use a 5-node Hadoop cluster (one master and four slaves) to run all data analysis workloads. The nodes in our Hadoop cluster are connected through 1 Gb ethernet network. Each node has two Intel Xeon E5645 (Westmere) processors and 32 GB memory. A Xeon E5645 processor includes six physical out-of-order cores with speculative pipelines. Each core has private L1 and L2 caches, and all cores share the L3 cache. Table III lists the important

Table III
DETAILS OF HARDWARE CONFIGURATIONS.

CPU Type	Intel (®)Xeon E5645
# Cores	6 cores@2.4G
# threads	12 threads
#Sockets	2
ITLB	4-way set associative, 64 entries
DTLB	4-way set associative, 64 entries
L2 TLB	4-way associative, 512 entries
L1 DCache	32KB, 8-way associative, 64 byte/line
L1 ICache	32KB, 4-way associative, 64 byte/line
L2 Cache	256 KB, 8-way associative, 64 byte/line
L3 Cache	12 MB, 16-way associative, 64 byte/line
Memory	32 GB , DDR3

hardware configurations of the processor.

B. Hadoop Cluster Environments

All the workloads are implemented on the Hadoop system, which is an open source MapReduce implementation. The version of Hadoop and JDK is 1.0.2 and 1.6.0, respectively. For data warehouse workloads, we use Hive of the 0.6 version. Each node runs Linux CentOS 5.5 with the 2.6.34 Linux kernel. Each slave node is configured with 24 map task slots and 12 reduce task slots. For each map and reduce task, we assigned 1 GB Java heap in order to achieve better performance.

C. Compared Benchmarks setups

In addition to data analysis workloads, we deployed several benchmark suites, including SPEC CPU2006, HPCC, and SPECweb 2005, CloudSuite—a scale-out benchmark suite for cloud computing [10], and compared them with data analysis workloads.

1) *Traditional benchmarks setups*: SPEC CPU2006: we run the official applications with the first reference input, reporting results averaged into two groups, integer benchmarks (*SPECINT*) and floating point benchmarks (*SPECFP*). The gcc which we used to compile the SPEC CPU is version 4.1.2.

HPCC: we deploy HPCC—a representative HPC benchmark suite. The HPCC version is 1.4. It has seven bench-

marks¹, including *HPL*, *STREAM*, *PTRANS*, *RandomAccess*, *DGEMM*, *FFT*, and *COMM*. We run each benchmark respectively.

SPECweb 2005: we run the bank application as the Web server on one node with 24 GB data set. We use distributed clients to generate the workloads, and the number of the total simultaneous sessions is 3000.

2) *CloudSuite Setups*: CloudSuite has six benchmarks, including one data analysis workload— *Naive Bayes*. We also choose *Naive Bayes* as one of the representative data analysis workloads with a larger data input set (147 GB). In [10], the data input size is only 4.5 GB.

We set up the other five benchmarks following the introduction on the CloudSuite web site [18].

Data Serving: we benchmark *Cassandra* 0.7.3 database with 30 million records. The request is generated by a YCSB [19] client with a 50:50 ratio of read to update.

Media Streaming: we use *Darwin* streaming server 6.0.6. We set 20 Java processes and issue 20 client threads by using the Faban driver [20] with GetMediumLow 70 and GetshortHi 30.

Software Testing: we use the *cloud9* execution engine, and run the *printf.bc* coreutils binary file.

Web Search: we benchmark a distributed *Nutch* 1.1 index server. The index and data segment size is 17, and 35 GB, respectively.

Web Serving: we characterize a front end of *Olio* server. We simulate 500 concurrent users to send requests with 30 seconds ramp-up time and 300 seconds steady state time.

3) *Data Analysis Workload Setups*: Table I presents the size of input data set and the instructions retired of each data analysis workload. The input data size varies from 147 to 187 GB. In comparison with that of CloudSuite described in [10], our approach are more pragmatic, and we deploy a larger data input that are stored in both memory and disk systems instead of completely storing data (only 4.5 GB for *Naive Bayes* in [10]) in memory. The number of instructions retired of the data analysis workloads ranges from thousand of billions to tens of thousands of billions, which indicate that those applications are not trivial ones.

D. Experimental Methodology

Modern superscalar Out-of-Order (OoO) processors prevent us from breaking down the execution time precisely due to overlapped work in the pipeline [10] [21] [22]. The retirement centric analysis is also difficult to account how the CPU cycles are used because the pipelines will continue executing instructions even though the instruction retirement is blocked [23]. So in this paper we focus on counting

¹*HPL* solves linear equations. *STREAM* is a simple synthetic benchmark, streaming access memory. *RandomAccess* updates (remote) memory randomly. *DGEMM* performs matrix multiplications. *FFT* performs discrete fourier transform. *COMM* is a set of tests to measure latency and bandwidth of the interconnection system.

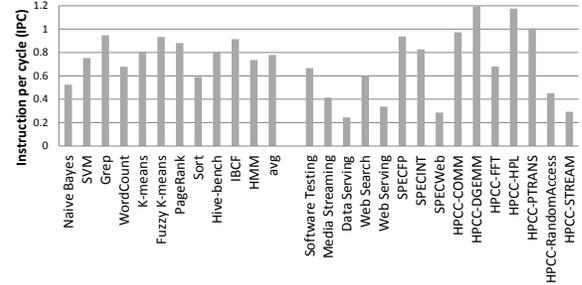


Figure 3. Instructions per cycle for each workload.

cycles stalled due to resource conflict, e.g. the reorder buffer full stall, which prevents new instructions from entering the pipelines.

We get the micro-architectural data by using hardware performance counters to measure the architectural events. In order to monitor micro-architectural events, a Xeon processor provides several performance event select MSRs (Model Specific Registers), which specify hardware events to be counted, and performance monitoring counter MSRs, which store results of performance monitoring events. We use Perf—a profiling tool for Linux 2.6+ based systems [24], to manipulate those MSRs by specifying the event numbers and corresponding unit masks. We collect about 20 events whose number and corresponding unit masks can be found in the Intel Developer’s Manual [25]. In addition, we access the *proc* file system to collect OS-level performance data, such as the number of disk writes.

We perform a ramp-up period for each application, and then start collecting the performance data. Different from the experiment methodology of CloudSuite, which only performs 180-second measurement, the performance data we collected cover the whole lifetime of each application, including map, shuffle, and reduce stages. We collect the data of all the four working nodes and report the mean value.

IV. RESULTS

We provide a detailed analysis of the inefficiencies of running data analysis workloads on modern OoO processors in the rest of this section.

A. Instructions Execution

Instruction per cycle (in short IPC) is used to measure instruction level parallelism, indicating how many instructions can execute simultaneously. Our processors have 6 cores, and each core can commit up to 4 instructions on each cycle in theory. However, for different workloads, IPC can be limited by pipeline stall and data or instructions dependencies.

Figure 3 shows IPC of each workload. The CloudSuite has six benchmarks, among which we report the Naive Bayes on the leftmost side, separated from the other five workloads (in

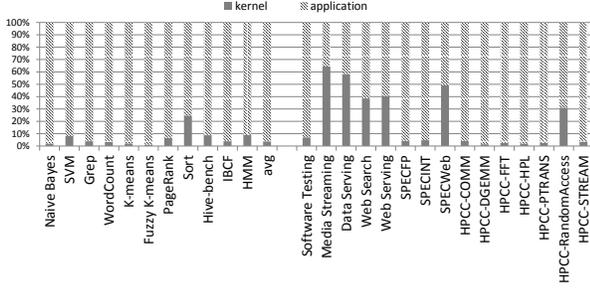


Figure 4. User and Kernel Instructions Breakdown.

the middle side), since *Naive Bayes* is also included into our eleven workloads.

The main workloads of CloudSuite (four among six) are service workloads: *Media Streaming*, *Data Services*, *Web Services*, and *Web Search*. From Figure 3, we can observe that service workloads, including four of CloudSuite and *SPECweb* has the lower IPC (all less than 0.6) in comparison with the other workloads, including our chosen data analysis workloads, *SPECFP*, *SPECINT*, and most of HPC workloads.

Most of data analysis workloads have middle-level IPC values, greater than that of the service workloads. The IPC of the eleven data analysis workloads ranges from 0.52 to 0.95 with an average value of 0.78. The avg bar in Figure 3 means the average IPC of the eleven data analysis workloads. *Naive Bayes* has the lowest IPC value among the eleven data analysis workloads. The IPCs of the HPC workloads have a large discrepancy among each workload since they are all micro-benchmark designed for measuring different aspects of systems. For example, *HPCC-HPL* and *HPCC-DGEMM* are computation-intensive, and hence have a higher IPC (close to 1.2). While *HPCC-STREAM* are designed to stream access memory, it has poor temporal locality, causing long-latency memory accesses, and hence it has lower IPCs (less than 0.5).

Figure 4 illustrates the retired instructions breakdown of each workload. We also notice that the service workloads (four of CloudSuite and *SPECweb*) execute a large percentage of kernel-mode instructions (greater than 40%), while most data analysis workloads execute a small percentage of kernel-mode instructions. The service workloads have higher percentages of kernel-mode instructions because serving a large amount of requests will result in a large number of network and disk activities.

Among the data analysis workloads, only *Sort* has a high proportion (about 24%) of kernel-mode instructions whereas on average the data analysis workloads only have about 4% instructions executed in kernel-mode. This is caused by the two unique characteristics of *Sort*. The first one is that different from most of the data analysis workloads, the input data size of *Sort* is equal to the output data size.

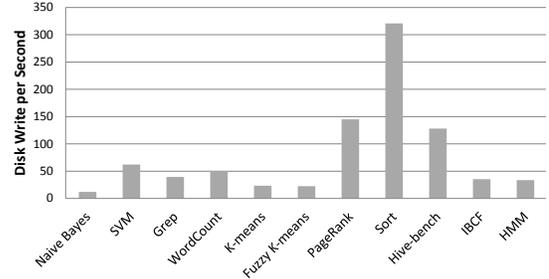


Figure 5. Disk Writes per Second.

So in each stage of the MapReduce job, the system will write large amount of output data to local disk or transfer a large amount of data over network. This characteristic makes *Sort* have more I/O operations than other workloads. The second unique characteristic is that *Sort* has simple computing logic, only comparing. So it can process a large amount data in a short time. Those characteristics let *Sort* involve more frequently I/O operations (disk and network). So in comparison with other data analysis workloads, *Sort* are more OS-intensive. Figure 5 depicts disk writes per second of each data analysis workload. We can find that *Sort* has the highest disk writes frequency. We also observed that network communication activities of *Sort* are also more frequent than that of the other data analysis workloads.

Among the HPC workloads, *RandomAccess* has a large percentage of kernel-mode instructions (about 31%). *RandomAccess* measures the rate of integer random updates of (remote) memory. An update is a read-modify-write operation on a table of 64-bit words, and it involves a large amount of *copy_user_generic_string* system calls. The other factors contributing a large percentage of kernel-mode instructions need further investigations.

Observations:

Data analysis workloads have higher IPC than that of services workloads, which are characterized by CloudSuite and traditional web server workloads, e.g., *SPECweb2005*, while lower than that of computation-intensive workloads, e.g., *HPC-HPL*, *HPC-DGEMM*. Meanwhile we also observe that the most of data analysis workloads involve less kernel-mode instructions than that of the services workloads.

B. Pipeline Behaviors

Modern processor implements dynamic execution using out of order and speculative engine. The whole processor architecture can be divided into two parts: including an in-order front end, which fetches, decodes, and issues instructions, and an out-of-order back end, which executes instructions and write data back to register files. A stall can happen in any part of the pipeline. In this paper we focus on the major pipeline stalls (*not exhausted*), including front end (instruction fetch), register allocation table (in short RAT), load-store buffers, reservation station (in short RS),

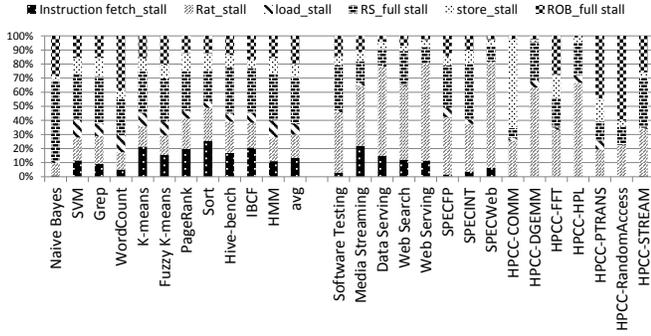


Figure 6. Pipeline Stall Break Down of Each Workload

and re-order buffer (in short *ROB*). For modern X86 architecture, front end will fetch instructions from L1 Instruction cache and then decode the CISC instructions into RISC-like instructions, which Intel calls micro-operation. *RAT* will change the registers used by the program into internal registers available. Load-store buffers are also known as memory order buffers, holding in-flight memory micro-operations (load and store), and they ensure that writes to memory take place in the right order. *RS* queues micro-operations until all source operands are ready. *ROB* track all micro-operations in-flight and make the out-of-order executed instructions retire in order.

Figure 6 presents those major stalls in pipelines for each workload including instruction fetch stalls, RAT stalls, load buffer full stalls, store buffer full stalls, RS full stalls, and ROB full stalls. We can get the blocked cycles of those kind of stalls mentioned above by using hardware performance counters. Different kinds of pipeline stalls may occur simultaneously, that is to say, the stall cycles may overlap. For example, when the back end is stalled due to *RS* full, the front end can also be stalled due to L1 instruction cache misses. So in Figure 6, we report the normalized values of the stalled cycles. We calculate the normalized value by using the following way: we sum up all the blocked cycles for all kinds of stalls as the total blocked cycles. Then we divide each kind of stall’s blocked cycles by the total blocked cycles as their percentage in Figure 6.

Different from HPC and SPEC CPU2006 workloads, the data analysis workloads and service workloads suffer from notable front end stalls, which are mainly caused by L1 instruction cache miss, ITLB (Instruction Translation Lookaside Buffer) miss or ITLB fault, reported in front-end performance data in Section IV-C. The notable front end stalls indicates the instruction fetch unit inefficiency. Our observation corroborates the previous work [10]. The front end inefficiency may caused by high-level languages and third-party libraries used by the data analysis and service workloads. The complicated framework and middleware will increase the binary size of the whole application even though they only implement a simple algorithm.

We also find that there are notable differences in terms of stalls breakdown between the data analysis workloads and service workloads (including four service workloads of CloudSuite and SPECWeb). The latter workloads own a large percentage of RAT stalls, which may be caused by partial register stall or register read port conflicts. While the data analysis workloads suffer from more RS stalls and ROB stalls, which are caused by limited RS and ROB entries. *RAT* and instruction fetch stalls occur before instruction entering the out-of-order part of the pipeline while the RS and ROB stalls occur at the out of order part of the pipeline. The service workloads (including *Media Streaming*, *Data Severing*, *Web Severing*, *Web Search* and *SPECweb*) have 60% RAT stalls and 13% instruction fetch stalls on average, whereas the data analysis workloads have about 37% RS full stalls and 20% ROB full stalls on average. So we can find that the data analysis workloads suffer more stalls in the out-of-order part of the pipeline, while the service workloads suffers more stalls in the in-order part of the pipeline. Further investigation is necessary to understand the root cause behind the differences between two kinds of workloads.

For the HPC workloads are composed of micro benchmarks and kernel programs, different programs focus on a specific aspect of the system. So their stall data vary dramatically from each other in Figure 6.

Implications:

Corroborating previous work [10], both the data analysis workloads and the service workloads suffer from notable front-end stalls (i.e. instruction fetch stall). The instruction fetch stalls means that the front end has to wait for fetching instructions, which may be caused by two factors: deep memory hierarchy with long latency in modern processor [10], and large binary size complicated by high-level language and third-party libraries.

However, we note the significant differences between the data analysis workloads and the service workloads in terms of stall breakdown: the data analysis workloads suffer more stalls in the out-of-order part of the pipeline, while the service workloads suffer more stalls before instructions entering the out-of-order part. This observation can give us some implications about how to alleviate the bottlenecks in pipeline, although one well known consequence is that right after of alleviating the bottleneck, the next bottleneck emerges [26].

C. Front-end Behaviors

The instruction-fetch stall will prevent core from making forward progress due to lack of instructions. Instruction cache and instruction Translation Look-aside Buffer (TLB) are two fundamental components, which must be accessed when fetching instructions from memory. Instruction cache is the place where the fetch unit directly get instructions. TLB stores page table entries (PTE), which are used to

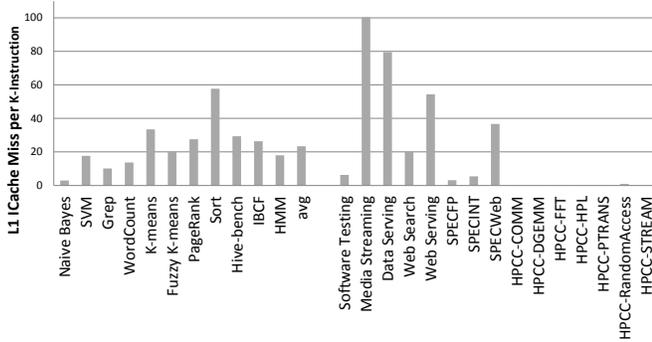


Figure 7. L1 Instruction Cache misses per thousand instructions.

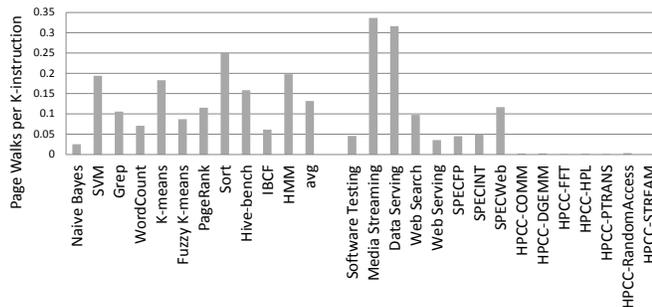


Figure 8. ITLB miss caused completed page walks per thousand instructions.

translate virtual addresses to physical addresses. If a TLB entry is found with a matching virtual page number, a TLB hit occurs and the processor can use the retrieved physical address to access memory. Otherwise there is a TLB miss, the processor has to look up the page table, which called a page walk. The page walk is an expensive operation.

Figure 7 and Figure 8 present the L1 instruction cache misses and the completed page walks caused by instruction TLB misses per thousand instructions, respectively. On average, the data analysis workloads generate about 23 L1 instruction cache misses per thousand instructions. They own higher L1 instruction cache misses than that of *SPECINT*, *SPECFFP*, and all the HPCC workloads. Most of the data analysis applications have less L1 instruction cache misses than that of the service workloads including *Media Streaming*, *Data Serving*, *Web Serving* and *SPECweb*. *Media streaming* has a larger instruction footprint and suffers from severe L1 Instruction cache misses, whose L1 Instruction cache misses are about three times more than the average of that of the data analysis workloads. Higher L1 instruction cache misses result in higher instruction fetch stalls as shown in Figure 6, indicating less efficiency of the front-end. For most of the others benchmarks, the L1 instruction cache misses are really very rare, especially the HPCC workloads, whose instruction footprint is relatively small.

Consistent with the performance trend of L1 instruction

cache misses, the data analysis workloads' completed page walks caused by instruction TLB miss are more frequently than that of *SPECINT*, *SPECFFP*, and all HPCC workloads. Some service workloads (*Media Streaming* and *Data Serving* workloads) have more completed page walks than that of the data analysis workloads. Page walks will cause a long latency instruction fetch stall, waiting for correct physical addresses so as to fetch instructions, and hence result in inefficiency of front end. Among the data analysis workloads, *Naive Bayes* is an exception with the smallest L1 instruction cache misses and completed page walks caused by instruction TLB misses, so it can not represent the spectrum of all data analysis workloads.

Implications:

Improving the L1 instruction cache and instruction TLB hit ratios can improve the performance of data analysis workloads, especially the service workloads. The third-party libraries and high-level languages used by datacenter workloads may enlarge the binary size of applications and further aggravate the inefficiency of instruction cache and TLB. So when writing the program (with the support of third-party libraries and high-level languages), the engineers should pay more attention to the code size.

D. Unified Cache and Data TLB Behaviors

The manufacturers of processors introduce a deep memory hierarchy to reduce the performance impacts of memory wall. Modern processors own three-level caches. A miss penalty of last-level cache can reach up to several hundred cycles in modern processors.

Figure 9 shows the L2 cache misses per thousand instructions. Figure 10 reports the ratio of L3 cache hits over L2 cache misses. This ratio can be calculated by using Equation 1. Please note that we do not analyze the L1 data cache statistics for the miss penalty can be hidden by the out-of-order cores [27].

$$ratio = \frac{L2\ cache\ misses - L3\ cache\ misses}{L2\ cache\ misses} \quad (1)$$

For most of the data analysis workloads, they have lower L2 cache misses (about 11 L2 cache misses per thousand instructions on average) than that of the service workloads (about 60 L2 cache misses per thousand instructions on average) while higher than that of the HPCC workloads. The L2 cache statistic indicates the data analysis workloads own better locality than the service workloads. The HPCC workloads have different localities as the official web site mentioned, which can explain the different cache behaviors among the HPCC workloads.

From Figure 10, we can find that for both the data analysis workloads and service workloads, the average ratio of L2 cache misses that are hit in L3 cache (85.5% for the data analysis workloads and 94.9% for the service workloads) is higher than that of the HPCC workloads. We can conclude

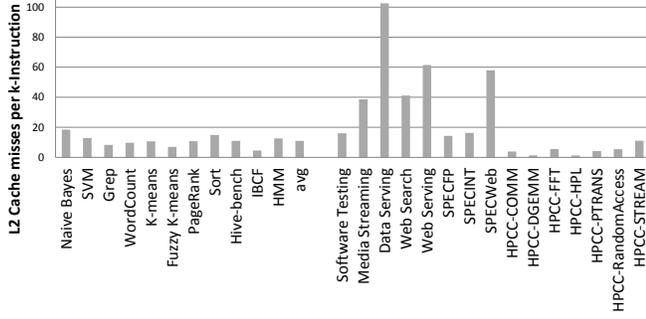


Figure 9. L2 cache misses per thousand instructions.

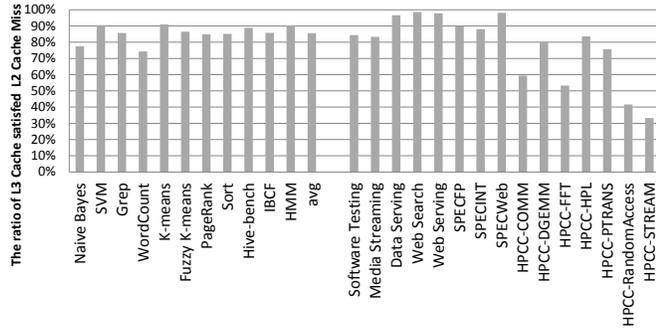


Figure 10. The ratio of L3 cache satisfying L2 cache misses.

that for most of the data analysis and service workloads, modern processor’s LLC is large enough to cache most of data missed from L2 cache.

Figure 11 shows the completed page walks caused by data TLB misses per thousand instructions. For most of the data analysis workloads with the exception of *Naive Bayes*, the completed page walks caused by data TLB misses are less than most of the service workloads and SPEC CPU2006 workloads (*SPECINT* and *SPECFP*), but higher than most of the HPC workloads with the exception of *HPC-RandomAccess* and *HPC-PTRANS*. That means the data locality of most of the data analysis workloads is much better than that of the service workloads.

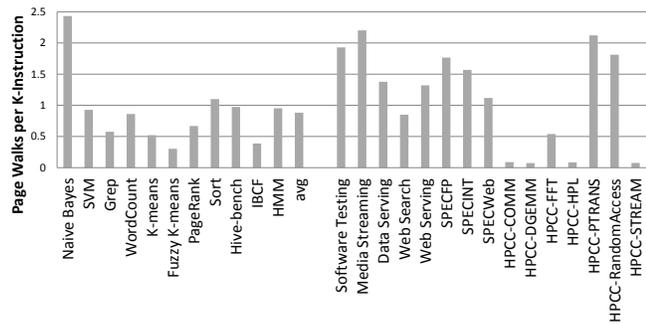


Figure 11. Completed Page Walks Caused by DTLB Misses per Thousand Instructions Retired.

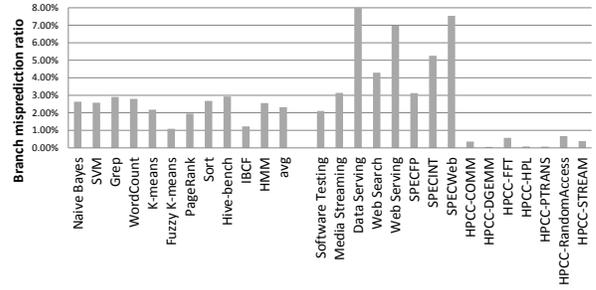


Figure 12. Branch Miss-prediction ratio.

Implications:

For the data analysis workloads, L2 cache is acceptably effective, and they have lower L2 cache misses than that of the service workloads, while higher than that of the HPC workloads. Meanwhile, for the data analysis and service workloads, most of L2 cache misses are hit in L3 cache, indicating L3 cache is pretty effective. Modern processors dedicate approximately half of the die area for caches, and hence optimizing the LLC capacity properly will improve the energy-efficiency of processor and save the die area. For the service workloads, our observation corroborate the previous work [10]: the L2 cache is ineffective.

E. Branch Prediction

The branch instruction prediction accuracy is one of the most important factor that directly affects the performance. Modern out-of-order processors introduce a functional unit (e.g. Branch Target Buffer) to predict the next branch to avoid pipeline stalls due to branches. If the predict is correct, the pipeline will continue. However, if a branch instruction is mispredicted, the pipeline must flush the wrong instructions and fetch the correct ones, which will cause at least a dozen of cycles’ penalty. So branch prediction is not a trivial issue in the pipeline.

Figure 12 presents the branch miss prediction ratios of each workload. We find that most of the data analysis workloads own a lower branch misprediction ratio in comparison with that of the service workloads and SPEC CPU workloads. The HPC workloads own very low misprediction ratios because the branch logic codes of the seven micro benchmarks are simple and the branch behaviors have great regularity. The low misprediction ratios of the data analysis workloads indicate that most of the branch instructions in the data analysis workloads have simple patterns. The simple patterns are conducive to BTB (Branch Target Buffer) to predict whether the next branch needs to jump or not. For the data analysis workloads, simple algorithms chosen for big data always beat better sophisticated algorithms[28], which may be the possible reason for their low misprediction ratios.

Implications:

Modern processors invest heavily in silicon real estate and algorithms for the branch prediction unit in order

to minimize the frequency and impact of wrong branch prediction. For the data analysis workload, the misprediction ratio is lower than most of the compared workloads, even for the CPU benchmark — SPECINT, which implies that the branch predictor of modern processor is good enough for the data analysis workloads. A simpler branch predictor may be preferred so as to save power and die area.

V. THE SUMMARY OF DCBENCH

Researchers in both academia and industry pay great attention to innovative data center computer systems and architecture. Since benchmarks, as the foundation of quantitative design approach, are used ubiquitously to evaluate the benefits of new designs and new systems [29], we released a benchmark suite named *DCBench* for datacenter computing with an open-source license on our project home page on <http://prof.ict.ac.cn/DCBench>.

According to our workload characterization work, the data analysis applications share many inherent characteristics, different from traditional server (SPECweb2005) and scale-out service workloads (four among six benchmarks in CloudSuite), so *DCBench* includes two different kinds of workloads: data analysis and service workloads. In our research work, we also notice the significant effects of different programming models, e. g., MPI vs. MapReduce, on the application behaviors, which is beyond the scope of this paper, so we also include the implementation of *DCBench* with different programming models on our homepage. Other important factors include OS [30] and VM executions, so we also provide some VM images on our homepage for downloading. We hope that *DCBench* is helpful for performing architecture and small-to-medium scale system researches for datacenter computing.

In addition to *DCBench*, we also release a big data benchmark suite—*BigDataBench* from Internet services [31], and a cloud computing benchmark suite—*CloudRank* [32]. The purpose of *BigDataBench* is for large-scale system and architecture researches. We believe that the focus of cloud computing is to consolidate different workloads on a datacenter, which provides elastic resource management. So the goal of *CloudRank* is to model complex usage scenarios of cloud computing for the purpose of capacity planning, system evaluation and researches.

VI. CONCLUSION AND FUTURE WORK

In this paper, after investigating most important application domains in terms of page views and daily visitors, we chosen eleven representative data analysis workloads and characterized their micro-architectural characteristics on the systems equipped with modern superscalar out-of-order processors by using hardware performance counters.

Our work shows that the data analysis workloads are significantly diverse in terms of both speedup performance and micro-architectural characteristics. In a word, only one

application is not enough to represent various categories of data analysis workloads.

Our study on the workloads reveals that the data analysis applications share many inherent characteristics, which place them in a different class from desktop, HPC, traditional server and scale-out service workloads, and accordingly we give several recommendations for architecture and system optimizations. Meanwhile, we also observe that the scale-out service workloads (four among six benchmarks in CloudSuite) share many similarity in terms of micro-architectural characteristics with that of the traditional server workload characterized by SPECweb 2005. We will investigate more workloads to confirm this observation.

ACKNOWLEDGMENT

We are very grateful to anonymous reviewers. This work is supported by the Chinese 973 project (Grant No.2011CB302502), the Hi-Tech Research and Development (863) Program of China (Grant No. 2011AA01A203, 2013AA01A213), the NSFC project (Grant No.60933003, 61202075), the BNSF project (Grant No.4133081) and the 242 project (Grant No.2012A95).

REFERENCES

- [1] J. Zhan, L. Zhang, N. Sun, L. Wang, Z. Jia, and C. Luo, "High volume throughput computing: Identifying and characterizing throughput oriented workloads in data centers," in *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*. IEEE, 2012, pp. 1712–1721.
- [2] L. Barroso and U. Hözl, "The datacenter as a computer: An introduction to the design of warehouse-scale machines," *Synthesis Lectures on Computer Architecture*, vol. 4, no. 1, pp. 1–108, 2009.
- [3] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. Sen Sarma, R. Murthy, and H. Liu, "Data warehousing and analytics infrastructure at facebook," in *Proceedings of the 2010 international conference on Management of data*. ACM, 2010, pp. 1013–1020.
- [4] <http://wiki.apache.org/hadoop/PoweredBy>.
- [5] J. Zhan, L. Wang, X. Li, W. Shi, C. Weng, W. Zhang, and X. Zang, "Cost-aware cooperative resource provisioning for heterogeneous workloads in data centers," *Computers, IEEE Transactions on*.
- [6] L. Wang, J. Zhan, W. Shi, and Y. Liang, "In cloud, can scientific communities benefit from the economies of scale?" *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, no. 2, pp. 296–303, 2012.
- [7] B. Sang, J. Zhan, G. Lu, H. Wang, D. Xu, L. Wang, Z. Zhang, and Z. Jia, "Precise, scalable, and online request tracing for multitier services of black boxes," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, no. 6, pp. 1159–1167, 2012.

