Understanding the Behavior of In-Memory Computing Workloads

Tao Jiang*†, Qianlong Zhang*, Rui Hou*, Lin Chai*, Sally A. Mckee‡, Zhen Jia*, and Ninghui Sun*
*SKL Computer Architecture, ICT, CAS, Beijing, China
†University of Chinese Academy of Sciences, Beijing, China
‡Chalmers University of Technology, Gothenburg, Sweden
{jiangtao, zhangqianlong, hourui, chailin, jiazhen, snh} @ict.ac.cn
mckee@chalmers.se

***Abstract*—The increasing demands of big data applications have led researchers and practitioners to turn to in-memory computing to speed processing. For instance, the Apache Spark framework stores intermediate results in memory to deliver good performance on iterative machine learning and interactive data analysis tasks. To the best of our knowledge, though, little work has been done to understand Spark's architectural and microarchitectural behaviors. Furthermore, although conventional commodity processors have been well optimized for traditional desktops and HPC, their effectiveness for Spark workloads remains to be studied.**

To shed some light on the effectiveness of conventional general-purpose processors on Spark workloads, we study their behavior in comparison to those of Hadoop, CloudSuite, SPEC CPU2006, TPC-C, and DesktopCloud. We evaluate the benchmarks on a 17-node Xeon cluster. Our performance results reveal that Spark workloads have significantly different characteristics from Hadoop and traditional HPC benchmarks. At the system level, Spark workloads have good memory bandwidth utilization (up to 50%), stable memory accesses, and high disk IO request frequency (200 per second). At the microarchitectural level, the cache and TLB are effective for Spark workloads, but the L2 cache miss rate is high. We hope this work yields insights for chip and datacenter system designers.

I. Introduction

The explosive growth of digital IT devices and Internet services has ushered in an era of big data. Off-line processing like that of Hadoop has heretofore been the dominant paradigm for big data processing. Recently, though, on-line big data processing has received increasing attention. One motivation is that many emerging Internet services require quick responses after efficient analytics on high-volume of datasets. For instance, a website recommendation system would expect on-line real-time analysis of customers willingness to buy based on their click behavior. In order to provide low-latency, in-memory computing is becoming the major approach for big data on-line, real-time processing. One typical case is Spark [29], a distributed computing framework similar to Hadoop. In contrast, though, Spark stores intermediate results in memory instead of on disk, and therefore it delivers lower latencies than Hadoop's storage-based approach.

From the hardware perspective, people want to learn whether existing systems and CPU architectures can still meet the challenges of big data workloads, and if not, what kind of optimizations or even revolutions are required. In order to answer such questions, it is necessary to analyze typical big data workloads to understand their microarchitectural and architectural behaviors.

Since Hadoop workloads contribute to the most important off-line processing scenarios, researchers have spent much effort investigating their runtime behaviors. The results of these studies illustrate some mismatches between existing hardware designs and Hadoop workload characteristics [17], [19]. For example, Ferdman et al. [17] report that the cache hierarchy of major commercial CPU architectures is inefficient for many Hadoop workloads, especially considering the high miss rates of both the I-Cache and the L2 cache.

However, there is very limited published work that investigates the microarchitectural and architectural behaviors of on-line, real-time big data processing workloads. One key question is whether server microarchitecture and memory system designed mostly for traditional applications can efficiently support these processing workloads. To try to answer this question, we choose Spark workloads and capture the behaviors of the microarchitecture and memory system via hardware performance counters and a custom-made memory trace collection device. Our work can help people gain a deeper understanding of the behavioral characteristics of Spark applications running on traditional data center servers and may provide meaningful insights to data center system designers.

To put the performance numbers of Spark workloads in perspective, we evaluate some traditional high-performance benchmarks and scale-out benchmarks, including SPEC CPU2006 [7], TPC-C [9], CloudSuite [17] and DesktopCloud [20]. In order to compare with Hadoop, we evaluate some Hadoop workloads with the same input datasets and algorithms as our Spark workloads. We run the benchmarks on a 17-node Xeon cluster. Compared to the high-performance benchmarks, we find that the Spark workloads exhibit significantly different characteristics when compared to the CloudSuite and DesktopCloud benchmarks. While these benchmarks also differ from Hadoop benchmarks, the differences in microarchitectural behavior are relatively smaller, as they share some characteristics. On the other hand, the difference in memory behavior compared to Hadoop is obvious. The major insights we derive are:

- **Spark workloads experience higher disk IO than**

978-1-4799-6454-3/14/$31.00 ©2014 IEEE

Hadoop workloads. To our surprise, Spark workloads have a high number of disk accesses per second. Since the workloads have large input and output datasets and Spark has shorter execution time than Hadoop, the average number of disk requests is larger. This phenomenon implies that Spark workloads might benefit from replacing hard disk with SSD.

- **Spark workloads have good memory bandwidth utilization and smooth memory accesses.** Spark's average bandwidth is about 40% of the peak bandwidth, while Hadoop only uses 15%. Burst memory accesses in Spark workloads constitute more than 90% of the memory bus traffic. The *burst bandwidth* is only 47% larger than the average bandwidth, while that of Hadoop is 198%, implying Spark's memory accesses patterns are more stable than Hadoop's. In addition, hot pages in some Spark applications account for up to 90% of the total memory accesses, even though they only occupy 10% of the total memory pages. Moreover, about 25% of Spark's memory accesses are to sequential addresses, compared to about 9% of Hadoop. These numbers indicate that high-bandwidth architectures are effective for in-memory computing frameworks.
- **The microarchitecture of general-purpose x86 server processors works well for Spark workloads.** Spark workloads running on general-purpose x86 processors have higher IPC (1, on average) than other benchmarks apart from SPEC CPU. The branch misprediction ratios of Spark workloads are lower than other benchmarks, which implies that the existing branch predictor is effective for iterative computing. The cache and TLB are effective, except that the L2 cache miss rate is high, implying that opportunities exist for cache hierarchy optimizations.

II. Related Work

To the best of our knowledge, there is no comprehensive or systematic research on the microarchitectural and memory access characteristics of Spark, although many researchers have studied the performance and resource utilization of data center computer systems. Those studies employ the traditional high-performance benchmarks, DesktopCloud, CloudSuite, virtualization workloads, and big data analytic benchmarks [25], [18], [26], [12], [13], [30].

Ferdman et al. [17] gather scale-out data center applications to create CloudSuite, which focuses on off-line analysis and web services. They find that scale-out workloads show very poor performance on existing computer systems. Jiang et al. [20] find that existing commodity processors are not as suitable for DesktopCloud workloads as they are for traditional desktop applications. Jia et al. [19] evaluate the microarchitectural characteristics of 11 representative data analysis workloads, showing that such applications share characteristics that distinguish them from DesktopCloud, HPC, and service workloads.

Researchers have also developed some representative big data benchmarks to analyze characteristics of data center applications. Wang et al. [25] develop BigDataBench and use it to show the specific microarchitectural behaviors of big data applications: compared to traditional applications, they have lower operation intensities and higher L2 cache miss ratios, and the LLC shows high efficiency for such applications. Furthermore, much research analyzes Hadoop behavior. Among them, Ren et al. [24] provide insight into performance and job characteristics via analyzing Hadoop traces derived from a 2000-node production Hadoop cluster in TaoBao, Inc.

III. Methodology

A. Hardware Platform

We perform our experiments on a 17-node x86 cluster. Each node has two Intel Xeon 2.40GHz E5645 processors and 64GB DDR3. To ensure the correctness of our microarchitectural analysis, we turn off simultaneous multithreading (SMT) in the BIOS. Every node has eight 7200 rpm SATA disks with a capacity of one Terabyte. Table I lists the hardware configuration of our experimental environment in detail.

TABLE I
Hardware Configuration

CPU	Intel Xeon E5645 @ 2.40GHz
# Sockets	2
# Cores per Socket	6
L1I	32 KB, 4-way
L1D	32 KB, 8-way
L2	256 KB, 8-way
LLC (L3)	12MB, 16-way
Memory	64 GB DDR3 1333MHz
BIOS Configuration	Hyper-Threading Disabled Turbo-Boost Disabled Hardware Prefetchers Enabled

B. Workloads

We analyze CloudSuite, SPEC CPU2006, TPC-C, and DesktopCloud benchmarks together with Spark and Hadoop to give comprehensive comparisons. For each experiment, we plot the mean and standard deviations for 10 trials.

1) Benchmarks of Spark and Hadoop: Since there is no authoritative benchmark on Spark and Hadoop, we carefully select five classes of benchmarks from AMPLab, ICT BigDataBench, and other official test cases included in the Spark and Hadoop installation packages. Each class is implemented on both Spark and Hadoop with the same algorithms and input datasets. Detailed information of these benchmarks is listed in Table II.

Naive Bayes is one of the most important algorithms from e-commerce. We select the version from BigDataBench, which focuses on multi-disciplinary research, including a few workloads covering search engine, social networks, and e-commerce. Naive Bayes can represent the domain of social

TABLE II
SPARK AND HADOOP INPUT DATASET

Benchmark name	Input Data Size
Naive Bayes	557GB
Grep	352.4GB
Hive and Shark	203.7GB
PageRank	4847571 nodes, 68993773 edges
Connected Components	4847571 nodes, 68993773 edges

networks and electronic commerce, and it is frequently used in spam recognition and web page classification. BigDataBench offers tools to generate input datasets for every benchmark.

Grep belongs to the domains of search engines, social networks, and e-commerce, and it is often used in log analysis, web information extraction, and fuzzy search applications. Dean and Ghemawat also use it in their work [15]. We again select the version from BigDataBench.

Hive and Shark [3], [16], [28] are a popular data warehousing system and an SQL engine based on Hadoop and Spark, respectively. To provide quantitative and qualitative performance comparisons of these analytic frameworks, AMPLab has released benchmarks to help generate understandable and reproducible results [8]. As presented, the input datasets are generated using Intel's Hadoop benchmark tools, and they consist of two SQL tables with page ranking and user visiting information embedded. The workload includes three SQL queries. The first query scans and filters the dataset, the second applies string parsing to each input tuple and then performs a high-cardinality aggregation, and the last joins a smaller table to a larger table and then sorts the results.

PageRank is used to determine the importance of a web page [22]. PageRank-like algorithms are frequently used in search engines. We select our PageRank benchmark from test cases offered by GraphX [27] and Giraph [2], which are integrated into Spark and Hadoop installation packages. Apache Giraph is an iterative graph processing system based on Apache Hadoop's MapReduce implementation, which comes from Google's Pregel graph processing system [21]. GraphX is a new parallel computing framework based on Spark. Compared to Giraph, GraphX joins the benefits of data-parallel and graph-parallel systems, and it performs graph computation jobs well.

ConnectedComponents is also from the test cases for Spark and Hadoop. It is common in the field of graph analysis. The input data for both PageRank and ConnectedComponents are downloaded from Stanford University website [6].

Spark and Hadoop are deployed on the same 17-node cluster. One node acts as master and is responsible for task scheduling, and the other 16 nodes serve as workers. All nodes run CentOS 6.2 with the 2.6.32 kernel. The versions of Hadoop and Spark are 1.0.4 and 0.9.1, respectively. For analytic frameworks, we use Hive-0.12.0 and Shark-0.8.0. The JDK version is 1.7.0.

2) Compared Benchmarks: ***A) SPEC CPU2006*** benchmarks include integer and floating point programs executing with the reference input sets, and each execution is pinned to a specified processor core.

B) TPC-C is used with 40 warehouses. The ramp-up time is set to one minute, which is enough for the load to reach a steady state.

C) CloudSuite represents a set of applications that dominate today's data centers. CloudSuite version 1.0 consists of six benchmarks and, like BigDataBench, includes Naive Bayes. Ferdman et al.'s [17] input dataset is too small (4.5GB) for a 17-node cluster, so we choose the version from BigDataBench with an input dataset of 557GB. We run the other five benchmarks from CloudSuite with the native input sets. The input file for the *Data Serving* benchmark Cassandra [1] in CloudSuite is a 10GB Yahoo! Cloud Serving Benchmark (YCSB) dataset. For the *Media Streaming* benchmark, we set up 20 Java processes with 50 client threads in each process. For *Web Serving*, we use two nodes, with one as the web server and the other as the database server. For *Web Search* we set up one node as the index processing server with an 8GB dataset and a 3GB index set. For *Software Testing*, we install Cloud9 [14] on five nodes, with one serving as the load balancer and the others acting as workers.

D) DesktopCloud is one of the fastest growing segments of the cloud computing market, replacing traditional desktop computers with completely virtualized systems. The Xen-based guest OS of Domain0 is CentOS 5.5 with the 2.6.18 Linux kernel. Windows XP and CentOS 5.5 are installed for the other 14 Domains. To simulate real-world DesktopCloud utilization, we perform various operations such as watching videos, surfing web, anti-virus, browsing PDF, Office work, and web downloading on installed virtual machines.

C. Measurement Tools

Intel VTune Amplifier [5] is a commercial application used to analyze software performance for x86 based machines using hardware performance counters. We use the Vtune command line interface to profile our benchmarks and collect microarchitectural statistics. In addition, we use CentOS system tools to record utilization information for disk, memory, the CPU, and the network per second. To ensure accuracy, we clear the file-system cache and restart Hadoop and Spark before launching each test for all benchmarks.

However, using system tools can not give us all the memory information that we are interested in, such as the peak bandwidth of the memory references. We thus use Hyper Memory Trace Tracker (HMTT) [11], a hardware component situated between the DIMM and memory controller, to record all off-chip references. This helps us to better understand the in-memory behavior of Spark and Hadoop.

Given that we only have one suite of HMTT devices on hand, we set it up on a separate node not included in the 17-node cluster. We deploy all systems and make the node act as both master and workers, simultaneously. Every of memory access trace collection is launched after the benchmarks are

running in a stable state, and every collection lasts for 10 minutes.

IV. Evaluation and Analysis

This section presents the performance results of our experiments, including execution performance, memory access behavior, and microarchitectural behavior.

A. Execution Performance

1) Execution Time: Figure 1 shows the execution time of Spark and Hadoop benchmarks. The execution time of Hadoop benchmarks ranges from 2.7 times to 8.4 times those of the corresponding Spark benchmarks. This is due to the basic differences between the Spark and Hadoop frameworks: the former places and processes the datasets in memory whereas the latter frequently accesses disk. This observation corroborates previous work by Zeharia et al. [29].

2) Disk I/O: Figure 2 shows disk requests per second for Spark and Hadoop benchmarks. The frequency of reads/writes varies with application types. However, Spark ratios of read to write requests are similar to those of the corresponding Hadoop applications. ***To our surprise, disk requests per second for the Spark benchmarks are greater than for the Hadoop benchmarks, on average.*** This may be due to the fact that the Spark benchmarks run faster yet perform the same number of disk accesses for the same input and output datasets.

B. Memory Access Behavior

1) Bandwidth: In general, one of the most important standards by which to evaluate application memory requirements is the average bandwidth to access memory. However, memory bandwidth does not remain constant throughout execution. Thus, in order to better understand the memory access behaviors of Spark and Hadoop and possibly the differences between them, we evaluate their burst bandwidths. To calculate the average and burst bandwidth we sample the memory bandwidths every 1ms, and define burst bandwidth to be the average value of the top 10th percentile of the bandwidth samples, as in Bao et al. [11].

Figure 3 shows the average and burst bandwidths of all the Spark and Hadoop benchmarks. On average, the five Hadoop applications only use 15% of the peak bandwidth of 6.4GB/s (we limit the memory to 800MHz because that is HMTT's maximum sampling speed), while Spark's average bandwidth reaches about 40% of the peak bandwidth. This is 2.6 times that of Hadoop, indicating that Spark can make much better use of available memory bandwidth. In particular, the burst bandwidth of NaiveBayes on Spark reaches more than 80% of the peak bandwidth. Hadoop's bandwidth can exceed 198% of its average bandwidth, while Spark's burst bandwidth is only 47% higher. ***Hadoop's higher burst bandwidth reflects its uneven memory access behavior; Spark's memory access behavior is much more stable.***

2) Page Access Frequency: Memory page access frequency information is critical for optimizing application memory allocation and management. In our experiments, we collect the reference number of each memory page and perform some statistical analysis. Figure 4 shows that 80% of the memory requests access only 20% of the pages. ***Specifically, for some Spark applications, 90% of the memory requests access just 10% of the pages, which implies that the access locality is good.*** Giving special attention to managing these hot pages may improve system performance. Note that due to its streaming behavior, such a hot page distribution is not as clear for grep as for other benchmarks.

3) Burst Accesses: Figure 5 shows the cumulative distribution of the number of bursts of different sizes for all workloads. A burst size of one indicates transmitting one cacheline on the memory bus. In the following discussion, burst flows are traffic flows with burst sizes larger than one. On average, with about 60% of traffic flows on the memory bus being burst flows, the percentage of burst flows of Spark workloads is 50% higher than that of Hadoop. What is more, the figure also demonstrates that Spark workloads have larger burst sizes than Hadoop, since the size of almost all Hadoop burst requests is less than 16.

To further evaluate the memory access characteristics of Spark and Hadoop, we calculate the total memory bus traffic with different burst sizes. Figure 6 shows these cumulative distributions. For Spark workloads, about 90% of the bus traffic is bursts, while for Hadoop workloads, this is only about 70%. Based on analysis of the two figures, we find that ***Spark exhibits better memory bandwidth utilization***. Hadoop is limited by its larger proportion of non-burst memory requests.

4) Sequential Accesses: Next we analyze memory stalls as represented by L2 cache misses. We count the average number of cycles during which the L2 MSHRs are not empty. Figure 7 shows memory stall statistics for the Spark and Hadoop workloads. Although Spark has higher bandwidth utilization, it exhibits little difference from Hadoop with respect to its ratio of memory stalls, indicating that Spark's main loop puts little stress on the memory access module of the back-end of the pipeline.

To see why, we use HMTT to analyze the characteristics of the burst memory accesses. Figure 8 shows that ***Spark workloads have more memory requests with sequential addresses.*** Specifically, about 30% of the memory requests of GraphX workloads are sequential. In order to reduce the number of cache misses, many processors use hardware prefetchers to load cache lines before they are requested. Such mechanisms have been shown to perform well with traditional benchmarks [23]. We therefore speculate that frequently correct prefetches reduce the number of pipeline stalls caused by the load store unit.

C. Microarchitecture

1) IPC: Figure 9 shows the IPCs of the benchmarks we evaluate. The figure shows that the Spark benchmarks achieve

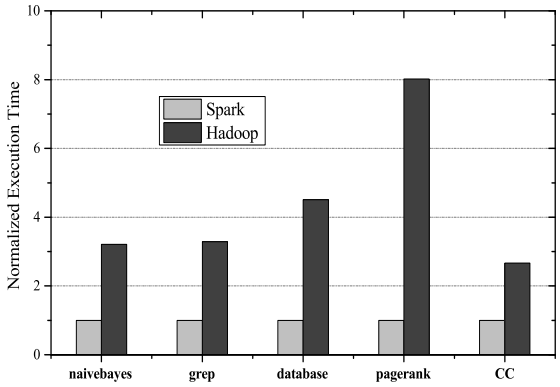

Fig. 1. Execution Time

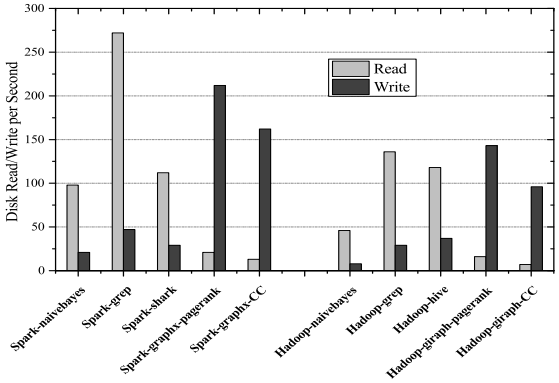

Fig. 2. Disk Accesses per Second

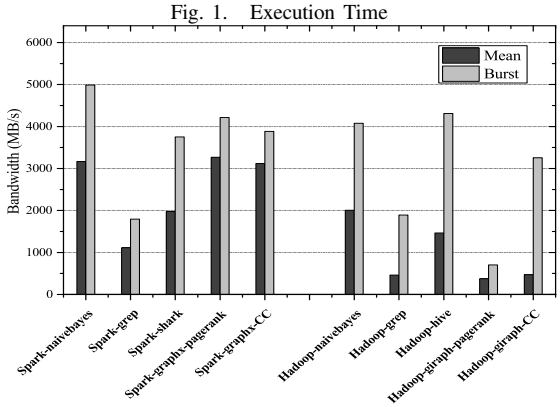

Fig. 3. Memory Bandwidth

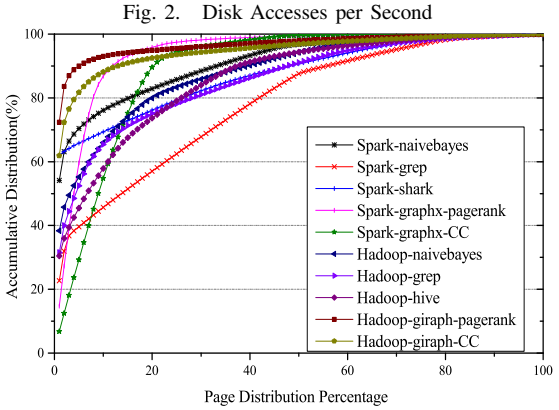

Fig. 4. Page Access Frequency

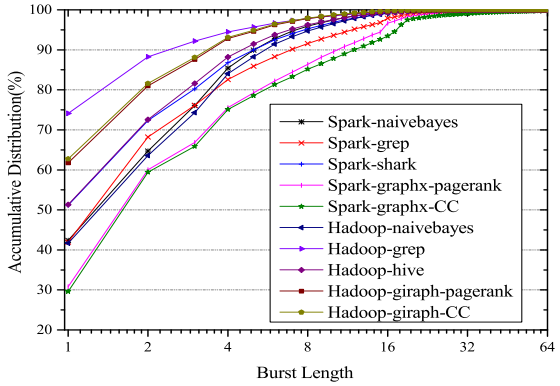

Fig. 5. Burst Memory Access Distribution

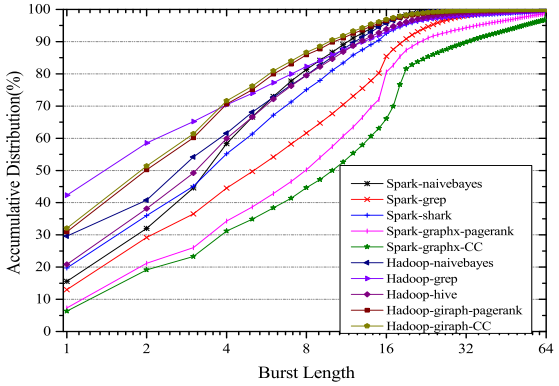

Fig. 6. Memory Bus Traffic Distribution

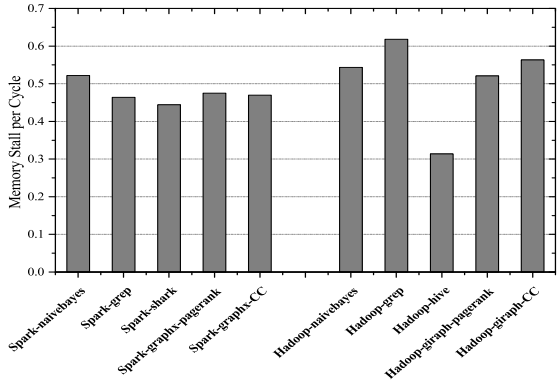

Fig. 7. Memory Stall Cycles

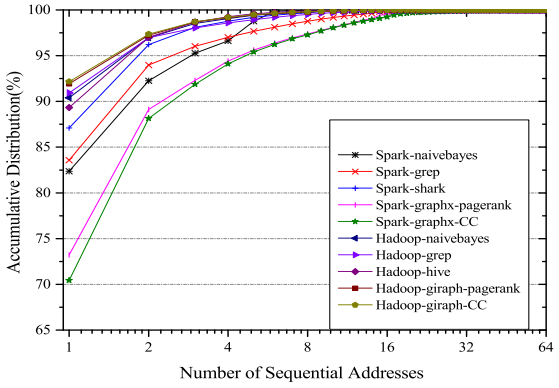

Fig. 8. Sequential Memory Accesses

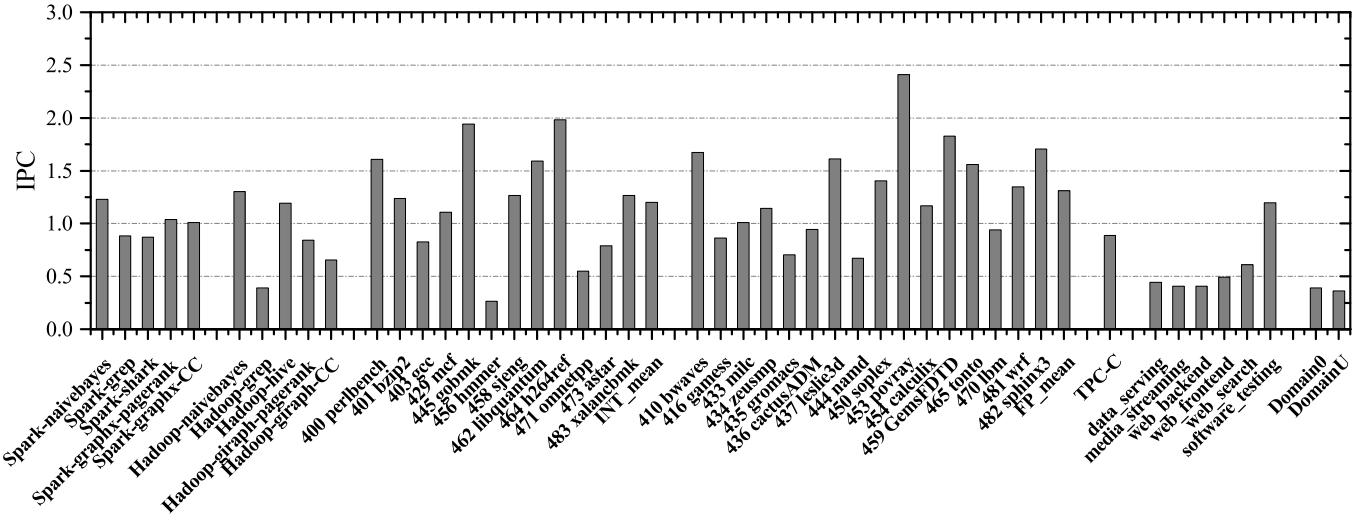

Fig. 9. IPC

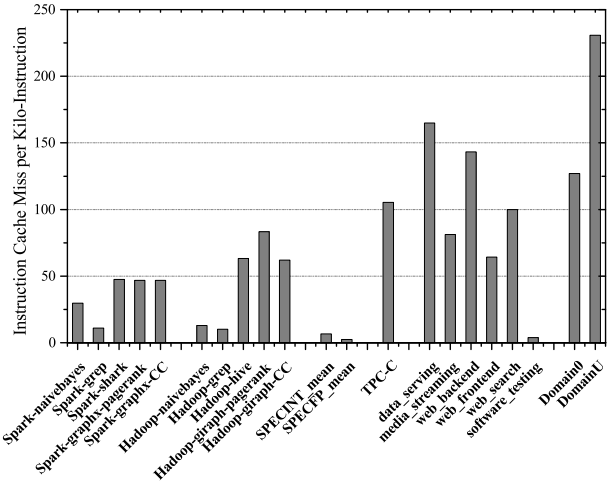

Fig. 10. L1I Miss per Kilo-Instruction

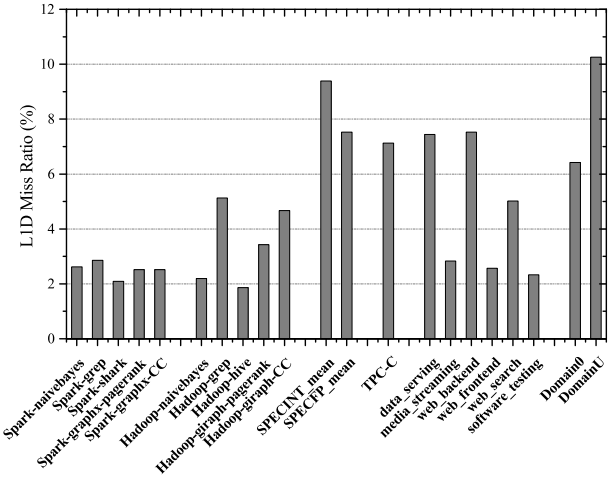

Fig. 11. L1D Miss Ratio

higher IPCs than the Hadoop benchmarks, the CloudSuite and DesktopCloud workloads. Composed of service applications with more interactive operations, we would expect the latter two workloads to have lower IPCs. On the other hand, compared to the traditional CPU-intensive benchmarks in SPEC CPU2006, Spark's IPCs are lower. This can be explained by the fact that Spark has more I/O operations. To analyze this more deeply, we collect detailed microarchitectural statistics for the I-Cache, D-Cache, TLB, and branch predictor, and we further investigate resource stalls.

2) L1I Cache: Instruction fetch efficiency directly affects pipeline utilization. Figure 10 shows L1 instruction misses per kilo-instruction (MPKI). It shows that the I-Cache miss rates of the Spark and Hadoop benchmarks are lower than those of the TPC-C, CloudSuite, and DesktopCloud workloads. For the data analysis workloads, Spark and Hadoop have fewer interactions than CloudSuite and DesktopCloud. However, they have higher MPKI than SPEC CPU2006. Because both Spark and Hadoop use Java Virtual Machine (JVM) to interpret and execute Java byte code on the physical machine, their extra I-Cache misses likely stem from the resulting larger software stacks. The Spark and Hadoop benchmarks experience more context switches among the different software layers.

3) L1D Cache: Figure 11 shows that the Spark and Hadoop benchmarks have lower L1D Cache miss rates than the other benchmarks, which indicates that they have better locality. Therefore, the L1D Cache is not a performance bottleneck: this indicates that the current L1 data cache works well for both Spark and Hadoop.

4) TLB: The translation look-aside buffer (TLB) is a common hardware structure to accelerate virtual-to-physical address translation.

A TLB miss triggers a process called page walk to load the associated translation into the TLB. Figure 12 presents

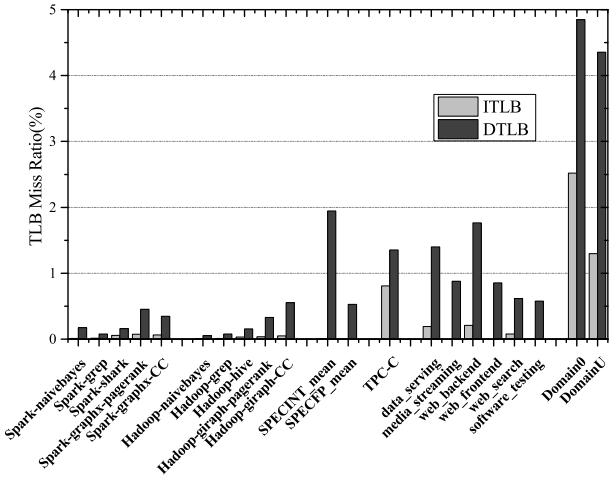

Fig. 12. TLB Miss Ratio

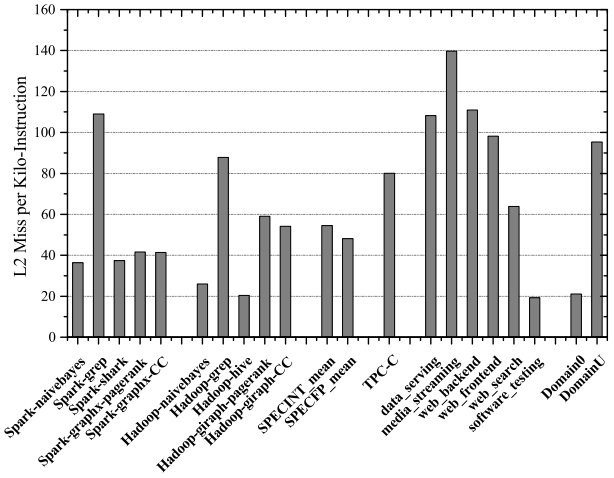

Fig. 14. L2 Miss per Kilo-Instruction

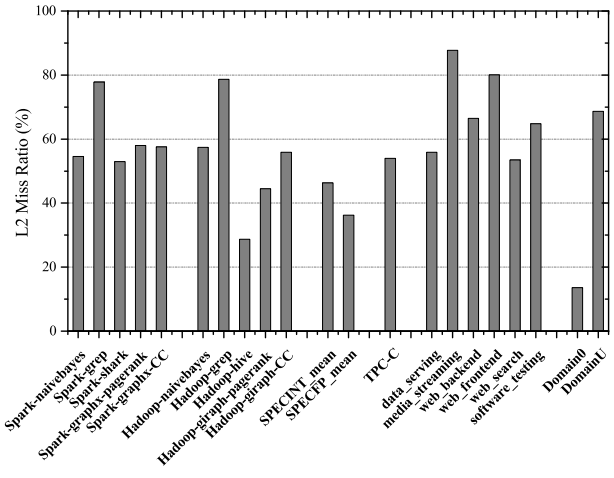

Fig. 13. L2 Miss Ratio

the ITLB and DTLB miss rates. The figure shows that both the Spark and Hadoop ITLB and DTLB miss rates are lower than those of TPC-C, CloudSuite, DesktopCloud and SPEC CPU2006. Babka et al. [10] show that the average TLB miss penalty is larger than the average L2 cache miss penalty, even with hardware page-table walks. These lower miss rates arise from the way Spark and Hadoop divide the full datasets into small pieces that are processed in parallel. Moreover, they maintain a one-to-one correspondence between jobs and CPU cores to avoid frequent context switches. Therefore, each core only processes one small-scale job at a time.

5) L2 Cache: Figure 13 shows that Spark and Hadoop suffer higher L2 miss rates than SPEC CPU2006 but lower than CloudSuite and DesktopCloud. Figure 14 shows that the Spark and Hadoop L2 MPKI are lower than those of the other benchmarks. These statistics indicate that the number of L2 cache references is relatively small. Therefore, the L2 cache is not a performance bottleneck for Spark and Hadoop, even though they suffer higher miss rates.

Among all the Spark and Hadoop benchmarks, the grep applications have the highest L2 cache miss rates and MPKI. Because grep must quickly process a large quantity of data, its main operation is a low-overhead comparison. Furthermore, grep streams these data sequentially, operating on each stream element only once.

6) Last Level Cache: The last level cache (LLC) is the last defense to mitigate the speed gap between the processor and the off-chip memory. Most server processors thus devote a large portion of chip area to the LLC. Figure 15 shows that all benchmarks except the grep application rarely suffer LLC misses. These results indicate that the LLC works well for almost all benchmarks, even though the L2 cache does not.

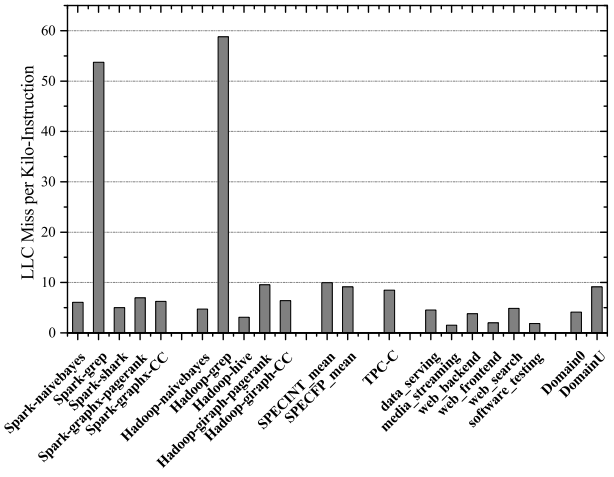

Fig. 15. LLC Miss per Kilo-Instruction

7) Branch Prediction: Branch predictors reduce stalls in the instruction pipeline and improve the performance of CPU. To reduce the frequency and cost of mispredictions, the branch predictors in modern processors tend to be quite sophisticated. Figure 16 shows that ***the Spark branch prediction miss rate is lower than those of other benchmarks***. Furthermore, the Hadoop branch prediction miss rate is also lower than most

of the others'. This reveals that the branch instructions of our Spark and Hadoop benchmarks are highly predictable. One possible reason is that the Spark and Hadoop benchmarks prefer simple algorithms. Our results indicate that the Intel branch predictor works well for these benchmarks.

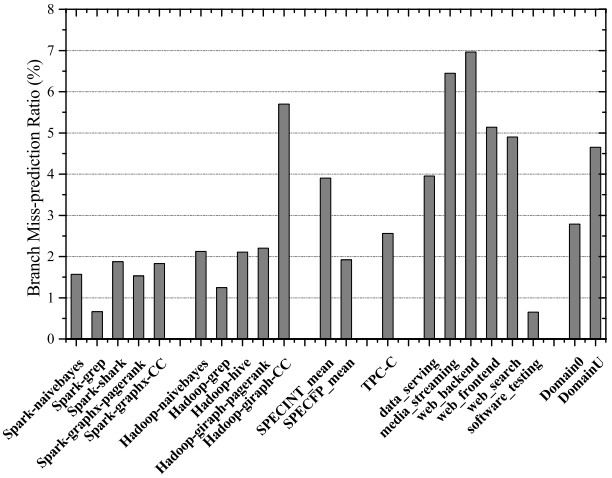

Fig. 16. Branch Miss-Prediction Ratio

8) Resource Stalls: Figure 17 shows the pipeline stall cycles caused by different operations, including register allocation stalls, reservation station full stalls, and reorder buffer full stalls. The resource stalls of the Spark and Hadoop benchmarks are higher than those of the TPC-C, CloudSuite, and DesktopCloud, but lower than those of the traditional CPU-intensive SPEC CPU2006 benchmarks. As expected, these results correspond to the computational intensities of the various benchmarks.

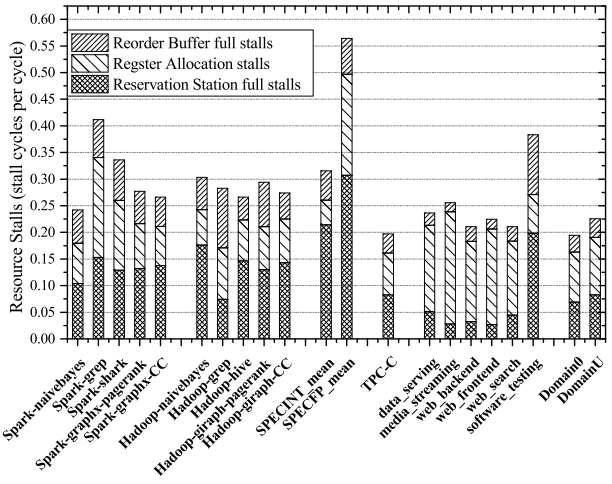

Fig. 17. Resource Stalls Breakdown

D. Discussion

Our experiments demonstrate that the memory access characteristics of the Spark and Hadoop workloads differ, in spite of their using the same algorithms and same input datasets. The Spark workloads could be further optimized to improve the efficiency of their memory referencing behavior.

On the chip level, previous work shows that general-purpose x86 server processors are not well suited to scale-out workloads [17]. In our experiments, the Spark benchmarks fare better than the Hadoop and scale-out benchmarks. Our Spark workloads exhibit high L2 cache miss rates, but low L1 cache and LLC miss rates. These results indicate that opportunities exist for cache hierarchy optimizations. On the memory level, the Spark's average bandwidth is about 40% of the peak bandwidth, while Hadoop only uses 15% of the peak bandwidth. The burst bandwidth of some Spark applications is up to 80% of the peak bandwidth, indicating that Spark workloads may benefit from memory bandwidth optimizations such as improving memory frequency and using devices such as the Hybrid Memory Cube (HMC) [4]. With respect to disk I/O, the access frequency of Spark workloads is much higher than Hadoop workloads, implying that Spark workloads might benefit from I/O optimizations.

V. Conclusion

In-memory computing is becoming one of the most popular approaches for real-time big data processing. It is thus important to better understand how the architecture and microarchitecture affect the performance of in-memory computing in order to determine whether existing systems can handle such workloads efficiently. In this paper, we use hardware performance counters and a custom-made memory trace collection device to analyze the behavior of Spark, Hadoop, SPEC CPU2006, TPC-C, CloudSuite, and DesktopCloud workloads. We gather a large set of performance statistics for all these workloads to show that with respect to many architectural features, the behavior of the Spark in-memory computing framework differs from Hadoop, scale-out service applications, DesktopCloud, and traditional high performance workloads.

We evaluate execution time, disk accesses per second, memory bandwidth utilization, page access frequency, and the burst and sequential access patterns of Spark and Hadoop benchmarks. Our results indicate that the Spark benchmarks use several times more memory bandwidth and have much higher burst behavior. Moreover, Spark workloads have more sequential burst accesses than Hadoop workloads. We also collect performance numbers on the L1 I and D caches, the TLB, the L2 cache, the LLC, and the branch predictor, and we track resource stalls. Our experimental results demonstrate that current Intel commodity processors are sufficiently efficient for in-memory computing.

VI. Acknowledgments

We thank our reviewers for their comments and suggestions. David Meisner provided valuable assistance in preparing the final copy of this manuscript. We also thank Licheng Chen for his help on evaluation with HMTT. This work was supported by the Strategic Priority Research Program of the Chinese

Academy of Sciences under grant No. XDA06010401, National Science Foundation of China under grant No. 61100010, No. 61402438 and No. 61402439.

REFERENCES

[1] Apache Cassandra. http://cassandra.apache.org.

[2] Apache Giraph. https://giraph.apache.org/.

[3] Apache Hive. http://hive.apache.org/.

[4] Hybrid Memory Cube. http://www.hybridmemorycube.org/.

[5] Intel VTune Amplifier XE 2013. https://software.intel.com/en-us/intel-vtune-amplifier-xe.

[6] LiveJournal Social Network. http://snap.stanford.edu/data/soc-LiveJournal1.html.

[7] SPEC CPU2006. http://www.spec.org/cpu2006/.

[8] The Big Data Benchmark of AMP Lab. https://amplab.cs.berkeley.edu/benchmark/.

[9] TPC-C.http://www.tpc.org/tpcc/default.asp.

[10] V. Babka and P. Tuma, "Investigating cache parameters of x86 family processors," in *Computer Performance Evaluation and Benchmarking*, January 2009, vol. 5419, pp. 77–96.

[11] Y. Bao, M. Chen, Y. Ruan, L. Liu, J. Fan, Q. Yuan, B. Song, , and J. Xu, "HMTT: A platform independent full-system memory trace monitoring system," in *International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, June 2008, pp. 229–240.

[12] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience (SPE)*, vol. 41, no. 1, pp. 23–50, January 2011.

[13] R. N. Calheiros, R. Ranjan, and R. Buyya, "Virtual machine provisioning based on analytical performance and QoS in cloud computing environments," in *International Conference on Parallel Processing (ICPP)*, September 2011, pp. 13–16.

[14] L. Ciortea, C. Zamfir, S. Bucur, V. Chipounov, and G. Candea, "Cloud9: A software testing service," *The ACM SIGOPS Operating System Review*, vol. 43, no. 4, pp. 5–10, January 2010.

[15] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Communications of ACM*, vol. 51, no. 1, pp. 107–113, January 2008.

[16] C. Engle, A. Lupher, R. Xin, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica, "Shark: Fast data analysis using coarse-grained distributed memory," in *International Conference on Management of Data (SIGMOD)*, May 2012, pp. 689–692.

[17] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the clouds: A study of emerging scale-out workloads on modern hardware," in *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, March 2012, pp. 37–48.

[18] N. E. Jerger, D. Vantrease, and M. Lipasti, "An evaluation of server consolidation workloads for multi-core designs," in *IEEE International Symposium on Workload Characterization (IISWC)*, September 2007, pp. 47–56.

[19] Z. Jia, L. Wang, J. Zhan, L. Zhang, and C. Luo, "Characterizing data analysis workloads in data centers," in *IEEE International Symposium on Workload Characterization (IISWC)*, July 2013, pp. 66–76.

[20] T. Jiang, R. Hou, L. Zhang, K. Zhang, L. Chen, M. Chen, and N. Sun, "Micro-architectural characterization of Desktop Cloud workloads," in *IEEE International Symposium on Workload Characterization (IISWC)*, November 2012, pp. 131–140.

[21] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: A system for large-scale graph processing," in *International Conference on Management of Data (SIGMOD)*, June 2010, pp. 135–146.

[22] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank citation ranking: Bringing order to the Web," Stanford InfoLab, Tech. Rep., November 1999.

[23] S. Palacharla and R. E. Kessler, "Evaluating stream buffers as a secondary cache replacement," in *International Symposium on Computer Architecture (ISCA)*, April 1994, pp. 24–33.

[24] Z. Ren, X. Xu, J. Wan, W. Shi, and M. Zhou, "Workload characterization on a production Hadoop cluster: A case study on Taobao," in *IEEE International Symposium on Workload Characterization (IISWC)*, November 2012, pp. 3–13.

[25] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang, C. Zheng, G. Lu, K. Zhan, X. Li, and B. Qiu, "BigDataBench: a big data benchmark suite from internet services," in *International Symposium On High Performance Computer Architecture (HPCA)*, Feburary 2014, pp. 488–499.

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

[27] R. S. Xin, D. Crankshaw, A. Dave, J. E. Gonzalez, M. J. Franklin, and I. Stoica, "GraphX: Unifying data-parallel and graph-parallel analytics," Computer Science Databases, Feburary 2014.

[28] R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica, "Shark: SQL and rich analytics at scale," in *International Conference on Management of Data (SIGMOD)*, June 2013, pp. 13–24.

[29] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Hot Topics in Cloud Computing (HotCloud)*, June 2010, pp. 10–10.

[30] C. Zheng, J. Zhan, Z. Jia, and L. Zhang, "Characterizing OS Behavior of scale-out data center workloads," in *Workshop on the Interaction amongst Virtualization, Operating Systems and Computer Architecture (WIVOSCA)*, June 2013.