Poster Paper
Impact Of Software Stack Version On Micro-architecture

Sraman Choudhury
PES University
Bangalore, Karnataka
sramakoo@gmail.com

Srikar Chundury
PES University
Bangalore, Karnataka
chundurysrikar@gmail.com

Subramaniam Kalambur and Dinkar Sitaram
PES University
Bangalore, Karnataka
subramaniamkv@pes.edu,dinkars@pes.edu

ABSTRACT

Open source Big Data frameworks such as Spark have been evolving quite rapidly. Many of the changes have addressed improvement in performance mainly focusing on the performance of the entire job executing on a distributed system. Past studies have reported micro-architectural performance characteristics of benchmarks based on these Big Data frameworks. Given the rapid changes to these frameworks, it is expected that some of these code changes will also have a micro-architectural impact. In this paper, we present a comparative study of performance of Apache Spark across two major revisions and demonstrate that there are micro-architectural differences in the way the applications use the hardware.

ACM Reference Format:
Sraman Choudhury, Srikar Chundury, and Subramaniam Kalambur and Dinkar Sitaram. 2019. Poster Paper Impact Of Software Stack Version On Micro-architecture. In *Tenth ACM/SPEC International Conference on Performance Engineering Companion (ICPE '19 Companion), April 7–11, 2019, Mumbai, India.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3302541.3311963

1 INTRODUCTION

Apache Spark[5] is a popular Big Data processing framework that distributes its computation over a large cluster of nodes. A number of workloads addressing Big Data applications have been introduced, notably BigDataBench[4], CloudSuite[2] and HiBench[3]. The factors that affect program behaviour include both the size of the data being processed and the features of the application. Wang et al [4] have demonstrated that with increasing datasize of these workloads the CPI(Cycles per instruction) decreases due to the increasing L3 cache misses while Dimitrov et al[1] have shown that operating with or without compression has a significant impact on CPI.

Our main contribution in this paper is to study how two major versions of Apache Spark behave on a modern multi-core processor. We demonstrate through an experimental approach, that there can be significant performance differences in the micro-architectural characteristics based on turning on or off certain features in the applications.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
ICPE '19 Companion, April 7–11, 2019, Mumbai, India
© 2019 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-6286-3/19/04.
https://doi.org/10.1145/3302541.3311963

We use an experimental approach by measuring micro-architectural counters when executing workloads under different versions of Apache Spark to measure the impact of individual features within Apache Spark that affect the performance.

2 EXPERIMENTAL METHODOLOGY

Quantifying the performance impact when Big Data applications requires detailed analysis of the processor for different versions of big data software and its components. We selected two different versions of Apache Spark- V1.3.0 and V2.2.0 for our comparison experiment. Additions to Spark V2.2 compared to V1.3 are the Tungsten Engine, Memory Manager, DAG Scheduler and some improvements in Error Recovery. We selected benchmarks from the BigDataBench suite and measured their performance while executing on a 2 Socket Intel Xeon Broadwell based machine with 32 cores and 64GB of DDR3 memory. We chose a set of **33** metrics have been chosen to account for hardware performance counters across these versions. Some of the metrics chosen were include Elapsed Time, Clock-ticks, Instructions Retired, CPI Rate, ICache Misses, Memory Bound, L1 Bound, Memory Bandwidth and Memory Latency as reported by Intel VTune.

2.1 Experiments Performed

2.1.1 Machine Setup. Performance metrics of an Intel Broadwell machine with Xeon CPU E5-2620 v4 have been collected keeping the max CPU frequency at 2.10GHz. The machine has 16 physical/32 logical split across 2 NUMA nodes, but the experiments were conducted on 8 logical cores on the same NUMA node. The machine has an L1 Cache of 512KB, an L2 Cache of2048 KB and the last level cache (LLC) size is 20MB. All other configurations like Scala version(2.11) JDK version (8) remain same for both the versions as the performance gap can be directly accounted due to software version change.

2.1.2 Environment Setup. We isolated the cores that will run the workload from the cores that will run the other processes on the system my affinitizing the existing processes to small set of cores. Changing the Core-Affinity of every process that is running, helps us to reserve a set physical core for experimenting and reduce variability across runs and isolate the application being measured. The ACPI driver is activated to use the user-space power governor as it allows setting of discreet CPU frequencies which do not change dynamically, instead of Intel's modern pstate driver; this further ensures that the variations across runs is minimized. Further, we disabled Hyper Threading to map all the metrics directly to its parent physical core though the experiments are conducted on

logical cores. We selected a few workloads, as enumerated in Table 1 from the BigDataBench suite[4] to perform a comparative study.

Micro Benchmarks	Word Count
	Sort
	Grep
Graph Benchmarks	Page Rank
	Connected Components
	K-means

Table 1: List of various benchmarks we chose from Big Data Bench

Metric	Spark V1.3	Spark V2.2	%
Time	3914.45	1482.31	62.13
cycles PTI	408.85	475.41	16.28
instructions	1.16E+13	9.90E+12	55.45
branches PTI	185.24	190.56	2.87
branch misses PTI	1.31	1.38	5.29
cache misses PTI	2.45	2.83	15.11
cache references PTI	8.32	10.42	25.25

Table 2: Performance metrics comparison of benchmark WordCount(50GB) on single core with different versions of Spark. Data is normalized per thousand instructions (PTI) for comparison

2.2 Preliminary Results

We executed various benchmarks listed in Table 1 with different versions of Apache Spark and collected the performance counters using Intel VTune. Figure 1 illustrates the comparison between the CPI across multiple benchmarks and seems to indicate Spark 2.2 delivering a lower CPI. However, a closer examination of the *wordcount* application as illustrated in Table 2 indicates that there is nearly a 55% reduction in the total number of instructions executed. We narrowed this to the Tungsten compiler that was introduced in Spark version 2.2 which is more efficient at code generation. The signature of the code however is similar in that both have approximately the same fraction of branches and branch misses. However, it is interesting to observe that the number of cache references and misses per thousand instructions has increased indicating an increased pressure on the cache.

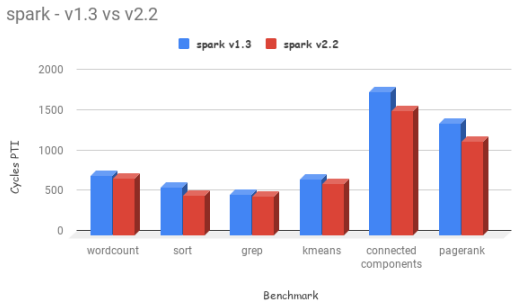

Figure 1: Overall Analysis - Spark

Figure 3 illustrates that the Tungsten compiler generates more efficient code. Also, note that the sensitivity of performance to datasizes without Tungsten is due to a reduction by 10% in the

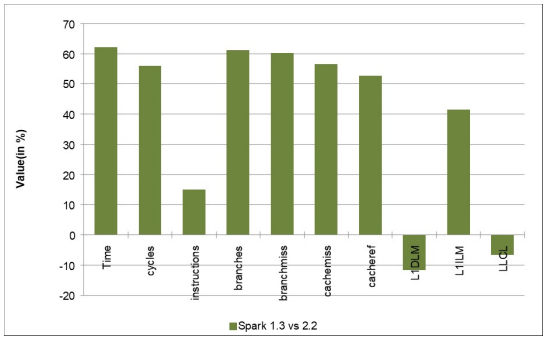

Figure 2: Spark Version Summary

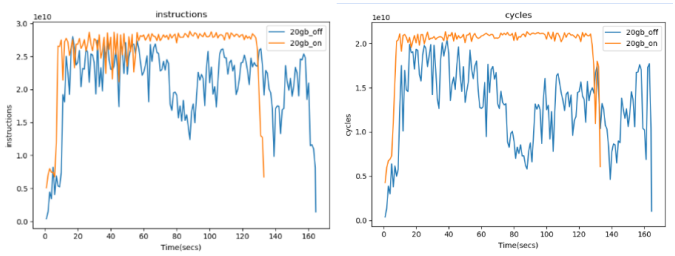

Figure 3: Instructions and Cycles
Tungsten ON vs Tungsten OFF - Benchmark: WordCount using DataFrames, Input Size: 20GB , Spark Version: 2.2

total number of page faults. Also, we observed that the IPC of the application dropped from 1.584 to 1.308 with Tungsten compiler on causes the application became more memory bound. This indicates to the hardware designer cache latency becomes a critical factor in their design.

ACKNOWLEDGMENTS

The authors would like to thank Intel,India for enabling them with a license to use Vtune amplifier tool for their studies.

REFERENCES

[1] Martin Dimitrov, Karthik Kumar, Patrick Lu, Vish Viswanathan, and Thomas Willhalm. 2013. Memory system characterization of big data workloads. In *Big Data, 2013 IEEE International Conference on*. IEEE, 15–22.

[2] Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. 2012. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. In *ACM SIGPLAN Notices*, Vol. 47. ACM, 37–48.

[3] Shengsheng Huang, Jie Huang, Yan Liu, Lan Yi, and Jinquan Dai. 2010. Hibench: A representative and comprehensive hadoop benchmark suite. In *Proc. ICDE Workshops*.

[4] Lei Wang, Jianfeng Zhan, Chunjie Luo, Yuqing Zhu, Qiang Yang, Yongqiang He, Wanling Gao, Zhen Jia, Yingjie Shi, Shujie Zhang, and others. 2014. Bigdatabench: A big data benchmark suite from internet services. In *High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on*. IEEE, 488–499.

[5] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster computing with working sets. *HotCloud* 10, 10-10 (2010), 95.