2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing

Main-Memory Requirements of Big Data Applications on Commodity Server Platform

Hosein Mohammadi Makrani, Setareh Rafatirad, Amir Houmansadr*, Houman Homayoun
Electrical and Computer Engineering Department, George Mason University, Fairfax, USA
College of Information and Computer Sciences, University of Massachusetts Amherst, USA*
{hmohamm8, srafatir, hhmoayou}@gmu.edu, amir@cs.umass.edu*

***Abstract*—The emergence of big data frameworks requires computational and memory resources that can naturally scale to manage massive amounts of diverse data. It is currently unclear whether big data frameworks such as Hadoop, Spark, and MPI will require high bandwidth and large capacity memory to cope with this change. The primary purpose of this study is to answer this question through empirical analysis of different memory configurations available for commodity server and to assess the impact of these configurations on the performance Hadoop and Spark frameworks, and MPI based applications. Our results show that neither DRAM capacity, frequency, nor the number of channels play a critical role on the performance of all studied Hadoop as well as most studied Spark applications. However, our results reveal that iterative tasks (e.g. machine learning) in Spark and MPI are benefiting from a high bandwidth and large capacity memory.**

***Keywords*—big data, memory, Hadoop, Spark, performance**

I. INTRODUCTION

Advances in various branches of technology – data sensing, data communication, data computation, and data storage – are driving an era of unprecedented innovation for information retrieval. The world of big data is constantly changing and producing substantial amounts of data that creates challenges to process it using existing solutions. To address this challenge, several frameworks such as Hadoop and Spark, based on cluster computing, have been proposed. The main characteristic of these new frameworks is their ability to process large-scale data-intensive applications on commodity hardware [1].

Big data analytics applications heavily rely on big-data-specific deep machine learning and data mining algorithms. They are running complex database software stack with significant interaction with I/O and OS, and exhibit high computational intensity and I/O intensity. In addition, unlike conventional CPU applications, big data applications combine a high data rate requirement with high computational power requirement, in particular for real-time and near-time performance constraints.

Three well-known parallel programming frameworks used by community are Hadoop, Spark, and MPI. Hadoop and Spark are two prominent frameworks for big data analytics. Spark has been developed to overcome the limitation of Hadoop on efficiently utilizing main memory. Both Hadoop and Spark use clusters of commodity hardware to process large datasets. MPI, a de facto industry standard for parallel programming on distributed memory systems, is also used for big data analytics [21].

While there are literatures on understanding the behavior of big data applications by characterizing them, most of prior works have focused on the CPU parameters such as core counts, core frequency, cache parameters, and network configuration or I/O implication with the assumption of the demand for using the fastest and largest main memory in the commodity hardware [3, 5, 12, 14, 15, 20, 24]. However, none of the previous works have studied the main memory subsystem parameters to characterize big data applications and the underlying frameworks.

The objective of this paper is to evaluate the effect of the memory subsystem on the performance of big data frameworks. To perform the memory subsystem analysis, we have investigated three configurable memory parameters including memory capacity, memory frequency, and number of memory channels, to determine how these parameters affect the performance and power consumption of big data applications. This analysis helps in making architectural decision such as what memory architecture to use to build a server for big data applications.

Our evaluation reveals that Hadoop applications do not require a high bandwidth-capacity memory subsystem to enhance the performance. Improving memory subsystem parameters beyond 1333 MHz Frequency and a single channel does not enhance Hadoop performance noticeably. Moreover, Hadoop framework does not require large capacity memory, since it stores all intermediates data on the storage rather than in the main memory. On the other hand, Spark and MPI applications can benefit from higher memory frequency and number of channels if the application is iterative such as machine learning algorithms. However, increasing the number of memory channels beyond two channels does not enhance the performance of those applications. This is an indication for lack of efficient memory allocation and management in both hardware (memory controller) as well as software stack. Furthermore, our results show that the memory usage of Spark framework is predictable that helps to do not overprovision the memory capacity for Spark based big data applications. On the other hand, MPI framework shows that its memory capacity requirement varies significantly across studied applications. This therefore indicates that applications implemented with MPI are requiring a large capacity memory to prevent from becoming a performance bottleneck.

Table 1: Studied workloads

Workload	wordcount	sort	grep	terasort	nweight	bayes	naïve bayes	kmeans	pagerank	aggregation	join	scan
Domain	micro kernel	micro kernel	micro kernel	micro kernel	graph analytics	e-commerce	e-commerce	machine learning	websearch	analytical query	analytical query	analytical query
Input type	text	data	text	data	graph	data	data	graph	data	data	data	data
Input size (huge)	1.1 T	178.8G	1.1 T	178.8G	17.6G	30.6G	30.6G	112.2G	16.8G	10.8G	10.8G	10.8G
Input size (large)	183.6G	29.8G	183.6G	29.8G	2.3G	5G	5G	18.7G	3.1G	1.8G	1.8G	1.8G
Input size (medium)	30.6G	3G	30.6G	3G	1.2G	1.6G	1.6G	3.7G	1.3G	1G	1G	1G
Framework	Hadoop, Spark, MPI	Hadoop, Spark, MPI	Hadoop, Spark, MPI	Hadoop, Spark	Spark	Hadoop, Spark	Hadoop, Spark, MPI	Hadoop, Spark, MPI	Hadoop, Spark	Hadoop	Hadoop	Hadoop
Suite	BigDataBench	BigDataBench	BigDataBench	HiBench	HiBench	HiBench	BigDataBench	BigDataBench	HiBench	HiBench	HiBench	HiBench

Table 2: MPI based Multimedia workloads from BigDataBench

Workload	BasicMPEG	DBN	Speech recognition	Image Segmentation	SIFT	Face Detection
Input type	DVD stream	Images	Audio	Images	Images	Images
Input size (huge)	24G	MNIST dataset	59G	62G	62G	62G

These findings are important as they help server designers to avoid over provisioning the memory subsystem for many of big data applications. Moreover, we found that the current storage systems are the main bottleneck for studied big data applications hence any further improvement of memory and CPU architecture without addressing the storage problem is a waste of money and energy. To the best of our knowledge this is the first work that looks beyond just the memory capacity to understand Hadoop, Spark and MPI based big data applications' memory behavior by analyzing the effect of memory frequency as well as number of memory channels on the performance as well of power consumption.

The remainder of this paper is organized as follows: Section 2 provides technical overview of the investigated workloads and the experimental setup. Results are presented in Section 3. Section 4 presents detailed discussions on the results. Section 5 describes related works. Finally, Section 6 concludes the paper.

II. Experimental Setup

In this section, we present our experimental system configurations and its setup. We describe our hardware platform and present experimental methodology. In our study, we used Hadoop MapReduce version 2.7.1, Spark version 2.1.0 in conjunction with Scala 2.11, and MPICH2 version 3.2 installed on Linux Ubuntu 14.04.

A. *Workloads*

For this study, we target various domains of applications namely that of microkernels, graph analytics, machine learning, E-commerce, social networks, search engines, and multimedia. We used BigDataBench [21] and HiBench [23] for the choice of big data benchmarking. We selected a diverse set of applications and frameworks to be representative of big data domain. More details of these workloads are provided in Table 1 and 2. The selected workloads have different characteristics such as high-level data graph and different input/output ratios. Some of them have unstructured data type and some others are graph based. Also, these workloads are popular in academia and are widely used in various studies.

B. *Hardware platform*

We carefully selected our experimental platform to investigate the micro-architectural effect on the performance of big data frameworks to understand whether our observations on memory subsystem behavior for big data remains valid for future architectures with enhanced microarchitecture parameters or not. This includes analyzing the results when increasing the core count, cache size and processor operating frequency. This is important, as the results will shed light on whether in future architectures larger number of cores, higher cache capacity and higher operating frequency change memory behavior of big data applications or not. Using the data collected from our experimental test setup, we will drive architectural conclusion on how these microarchitecture parameters are changing DRAM memory behavior and therefore impacting performance and energy-efficiency of big data applications. For running the workloads and monitoring statistics, we used a six-node standalone cluster with detailed characteristics presented in Table 3. To have a comprehensive experiment we used different SDRAM

Table 3: Hardware Platform

Hardware Type	Parameter	Value
Motherboard	Model	Intel S2600CP2
CPU	Model	Intel Xeon E5-2650 v2
	# Core	8
	# Threads	16(disabled)
	Base Frequency	2.6
	Turbo Frequency	3.4
	TDP	95
	L1 Cache	32 * 2 KB
	L2 Cache	256 KB
	L3 Cache	20 MB
	Memory Type Support	DDR3 800/1000/1333/1600/1867
	Maximum Memory Bandwidth	59.7 GB/S
	Max Memory Channels supported	4
Disk (SSD)	Model	HyperX FURY
	Capacity	480 GB
	Speed	500 MB/S
Disk (HDD)	Model	Seagate
	Capacity	500GB
	Speed	7200 RPM
Network Interface Card	Model	ST1000SPEXD4
	Speed	1000 Mbps

Table 4: Memory modules' part numbers and parameters studied in this work

DDR3	4 GB	8 GB	16 GB	32 GB
1333 MHz	D51264J90S	KVR13R9D8/8	KVR13R9D4/16	---
1600 MHz	D51272K111S8	D1G72K111S	D2G72K111	---
1867 MHz	KVR18R13S8/4	D1G72L131	D2G72L131	KVR18L13Q4/32

memory modules, shown in Table 4. All modules are provided from the same vendor. While network overhead in general is influencing the performance of studied applications and therefore the characterization results, for big data applications, as shown in a recent work [27], a modern high speed network introduces only a small 2% performance beneefit. We therefore used a high speed 1 Gbit/s network to avoid making it a performance bottleneck for this study. Our NICs have two ports and we used one of them per node for this study. Throughout this paper we will present the results based on high speed SSD disk.

C. Methodology

The experimental methodology of this paper is focused on understanding how big data frameworks are utilizing main memory.

1) Data collection: We used Intel Performance Counter Monitor tool (PCM) [22] to understand hardware (memory and processor) behavior. The performance counter data are collected for the entire run of each application, those counters were used to get the number of Bytes read or written by memory controller to calculate the memory bandwidth. We collect OS-level performance information with DSTAT tool—a profiling tool for Linux based systems by specifying the event under study. Some of the metrics that we used for this study are memory footprint, L2, and Last Level Cache (LLC) hits ratio, instruction per cycle (IPC), core C0 state residency, and power consumption. For power measurement, we used PCM-power utility [22], which provides the detailed power consumption of each socket and DRAM.

2) Parameter tuning: For both Hadoop and Spark frameworks, it is important to set the number of mapper and reducer slots appropriately to maximize the performance. Based on the result of [26], the maximum number of mappers running concurrently on the system to maximize performance should be equal to the total number of available CPU cores in the system. Therefore, for each experiment, we set the number of mappers equal to the total number of cores. We also follow same approach for the number of parallel tasks in MPI.

A recent work has shown that among all tuning parameters in a MapReduce framework, HDFS block size is most influential on the performance [25]. HDFS Block size has a direct relation to the number of parallel tasks (in Spark and Hadoop), as shown in EQ. (1).

$$\textit{Number of Tasks} = \textit{Input Size} \,/\, \textit{Block Size} \qquad \text{EQ. (1)}$$

In the above equation, the input size is the size of data that is distributed among nodes. The block size is the

Table 5: HDFS block size tuning

Application class	$\mathit{Input\ size} / \mathit{Number\ of\ nodes} \times \mathit{number\ of\ cores\ per\ node}$			
	< 64 MB	**< 512 MB**	**< 4 GB**	**> 4 GB**
CPU intensive	32 MB	64 MB	128 MB	256 MB
I/O intensive	64 MB	256 MB	512 MB	1 GB
Iterative tasks	64 MB	128 MB	256 MB	512 MB

amount of data that is transferred among nodes. Hence, block size has impact on the network traffic and its usage. Therefore, we first evaluate how changing this parameter affects the performance of the system. We studied a broad range of HDFS block sizes varying from 32 MB to 1GB when the main memory capacity is 64 GB and it has the highest frequency and number of channels. Table 5 demonstrates the best HDFS configuration for maximizing the performance in both Hadoop and Spark frameworks based on the ratio of Input data size to the total number of available processing cores, and the application class. The rest of the experiments presented in this paper are based on Table 5 configuration. We will present the classification of applications into CPU-intensive, I/O-intensive, and iterative tasks in next section. Our tuning methodology guarantees to put the highest pressure on memory subsystem.

III. Results

Our experimental results are presented in this section. First, we present the memory analysis of the studied workloads. We present how performance of studied workloads is sensitive to memory capacity, frequency and number of channels. Then, we provide results of architectural implication of processor parameters on big data frameworks and memory requirements. All performance metrics such as execution time, CPU active state residency, LLC and L2 hit ratio are discussed in this section. We also discuss the impact of storage system, size of input data, and cluster size on memory subsystem. In addition, we present the power analysis results. This is to help finding out which memory configuration is a better choice for energy-efficient big data processing.

A. Memory analysis

In this section, we present a comprehensive discussion on memory analysis results to help better understanding the memory requirements of big data frameworks.

1) Memory channels implication: The off-chip peak memory bandwidth equation is shown in EQ. (2).

$$\textit{Bandwidth} = \textit{Channels} \times \textit{Frequency} \times \textit{Width} \qquad \text{EQ. (2)}$$

We observe in Figure 1 that increasing the number of channels does not have significant effect on the execution time (on average 9%), except for K-means and Nweight in Spark, and for Image segmentation in MPI framework (All of them are iterative tasks). Figure 2 provides more insights to explain this exceptional behavior. This figure demonstrates the memory bandwidth usage of each workload. K-means, Image Segmentation, and Nweight bandwidth usages are shown to be substantially higher than other workloads. Hence providing more bandwidth decreases their execution time. By increasing the number of channels from 1 to 4, the gain is found to be 38%.

2) Memory frequency implication: As results in Figure 3 shows, similarly we don't observe any improvement of bandwidth usage or execution time by increasing memory frequency (from 1333 MHz to 1866 MHz) for most of applications. Note that increasing the frequency from 1333 MHz to 1866 MHz translates to almost 40% increase in the

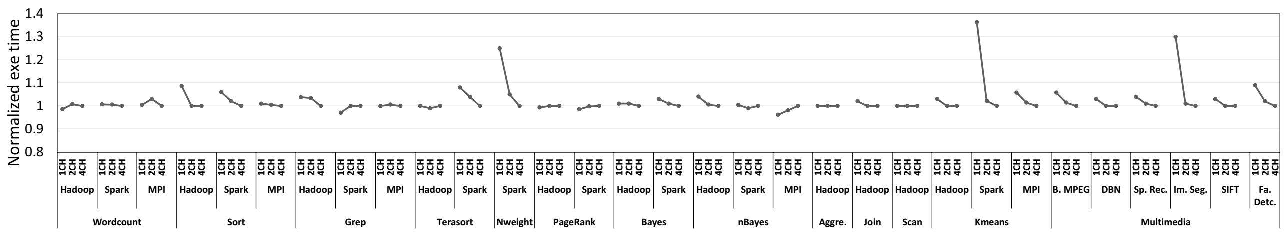
Figure 1: Effect of memory channel on the execution time (Normalized to 4CH)

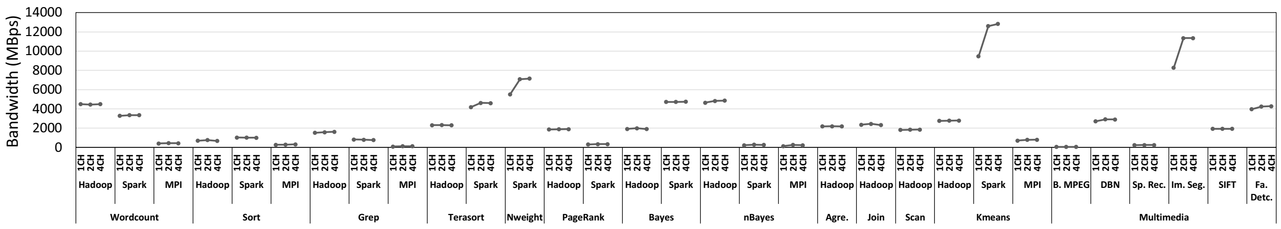
Figure 2: Impact of memory channel on bandwidth usage

bandwidth. Previous section showed that 400% (4X) increase in the bandwidth resulted by increasing the number of channels from 1 to 4 can gain only 9% performance benefit. Therefore, it is clear why a 40% increase in the bandwidth, because of increasing the memory frequency, cannot increase the performance noticeably. This finding may mislead to use the lowest memory frequency for big data applications. Based on EQ. (3), read latency of DRAM depends on the memory frequency.

Read latency = 2 × (CL / Frequency)   EQ. (3)

However, for DDRx (e.g. DDR3), this latency is set fixed by the manufacturer with controlling CAS latency (CL). This means two memory modules with different frequency (1333 MHz and 1866 MHz) and different CAS Latency (9 and 13) can have the same read latency of 13.5 ns, but provide different bandwidth per channel (10.66 GB/s and 14.93 GB/s). Hence, as along as reduction of frequency does not change the read latency, it is recommended to reduce DRAM frequency for most of big data applications unless the application is memory intensive. Later in this paper we will discuss memory sensitivity of studied applications.

3) DRAM capacity implication: To investigate the effect of memory capacity on the performance of big data applications, we run all workloads with 7 different memory capacities per node. During our experiments, Spark workloads encountered an error when running on a 4GB memory capacity per node due to lack of memory space for the Java heap. Hence, the experiment of Spark workloads is performed with at least 8 GB of memory. Based on our observation, we found that only MPI workloads have an unpredictable memory capacity usage. In fact, a large memory capacity has no significant effect on the performance of studied Hadoop and Spark workloads. In our experiments, the memory capacity usage of studied Hadoop and Spark never exceeded 4 GB, and 8 GB on each node respectively. Moreover, most of studied applications showed similar memory capacity usage. However, MPI based applications show different behavior. Hadoop applications do not require high capacity memory because Hadoop stores all intermediate values generated by map tasks on the storage. Hence, regardless of the number of map tasks or input size, the memory usage remains almost the same. Spark uses RDD to cache intermediate values in memory. Hence, by increasing the number of map tasks to run on a node, the memory usage increases. Therefore, by knowing the number of map tasks assigned to a node and the amount of intermediate values generated by each task, the maximum memory usage per node of Spark applications is predictable. To better understand the impact of memory capacity on the performance, we have provided the average normalized execution time of these three big data frameworks in Figure 4 (Normalized to 64 GB). To illustrate how these frameworks utilize DRAM capacity we present K-means memory usage on 3 different frameworks in Figure 5.

To explain the behavior of MPI, Hadoop, and Spark with respect to the memory capacity, we need to first understand how the operating system works when running these frameworks. Modern operating systems like Linux cache disk accesses by default when there is enough free space in RAM. The user cannot disable this mechanism manually in Linux. When the memory capacity is small, the available space for caching is low. Therefore, the system needs to frequently drop the old accesses and cache new ones. This process demands intensive memory management. However, increasing the memory capacity reduces the number of cache content droppings and reduces the intensity of memory management. Therefore, we observe that by increasing the memory capacity, the overhead of memory management of Linux is reduced.

As we can see in Spark workloads, the problem of disk caching is magnified when the memory footprint of workloads increases. For Spark workloads, when the memory capacity is 8 GB, due to large footprint of workloads, there is not enough free space and Linux therefore does not cache disk accesses. Hence, we don't observe caching overhead. Despite the reduced caching overhead, the memory capacity is still not big enough to

Table 6: Impact of the size of input data on memory usage

Framework	Hadoop			Spark			MPI		
Input size	**medium**	**large**	**huge**	**medium**	**large**	**huge**	**medium**	**large**	**huge**
Average memory capacity usage (MB)	3535	3486	3549	7365	7570	7662	2970	3168	6572
Standard deviation	217	224	236	54	62	71	1926	2178	4346
Max. memory Capacity usage (MB)	3843	3887	3860	7520	7691	7734	7405	7729	12650
Ave. Memory Bandwidth usage (MBpS)	2569	2683	2419	3508	3374	3288	713	680	523

contain workload's footprint and other applications' space (OS kernel and Spark services); therefore, this results in increased execution time. When the memory capacity is increased Linux starts to cache disk accesses. Therefore, due to caching overhead the average normalized execution time of Spark's workloads remains higher than when memory capacity is 8 GB. When memory capacity increases to 24 GB, the caching overhead of Spark workload is reduced. This is also the case for Hadoop and MPI applications.

4) Input size implication on memory capacity usage: Today the paradigm has been shifted and new MapReduce processing frameworks such as Hadoop and Spark are emerging that use disk as storage and rely on a cluster of servers to process data in a distributed manner. The ability of MapReduce frameworks is that each map task processes one block of data on HDFS at a time. Hence, this relieves the pressure of large input data on the memory subsystem. Therefore, regardless of input size, the memory subsystem usage remains almost constant in these frameworks. We have performed experiments with 3 sets of input data (medium, large, huge). However, all results presented in this paper are based on huge data size. Table 6 shows the memory usage of big data frameworks with varied sizes of input data. As it shows, the standard deviations of Hadoop and Spark memory usage are very low (less than 250 MB) and regardless of input data size, the average memory usages and maximum memory usages are close. However, this is different for MPI applications. Because MPI is not MapReduce based and does not use HDFS, the standard deviations are large and maximum memory usage is varied by input size and application type.

5) Input size implication on memory bandwidth usage: Another parameter that can be affected by the size of input data is the memory bandwidth usage. Our results presented in table 6 reveal that the size of input data does not noticeably change the memory behavior of big data frameworks. Because the memory bandwidth usage depends to the cache miss ratio of application (further we will discuss it in detail). Also cache behavior of application mostly depends to the application algorithm. Consequently, by increasing the size of input, the cache hit ratio remains almost the same. Therefore, while increasing the input size increases the job completion time, the DRAM bandwidth requirements of applications do not change noticeably.

B. Architectural analysis

As we discussed, several parameters, such as CPU frequency, number of cores per CPU, and cache hierarchy are studied in this paper to characterize big data frameworks. The micro-architectural analysis helps us to understand whether in future architectures with higher number of cores, larger cache capacity running at higher operating frequency, the observations regarding big data applications requirements remain valid or not.

1) Core frequency implication: Figure 6 shows that big data workloads behave in two distinct ways. The execution time of the first group is decreased linearly by increasing the core frequency. The second group's execution time does not drop significantly by increasing the CPU frequency, particularly when changing frequency from 1.9 GHz to 2.6 GHz. These two trends indicate that big data workloads have distinct behaviors of being either CPU bound or I/O bound. This conclusion further advocated by the results in Figure 7. This Figure proves sort, grep, PageRank, and scan

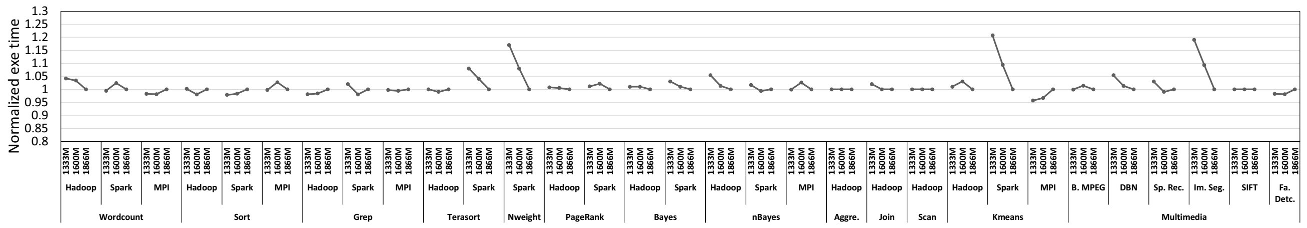

Figure 3: Effect of memory frequency on the execution time (Normalized to 1866M)

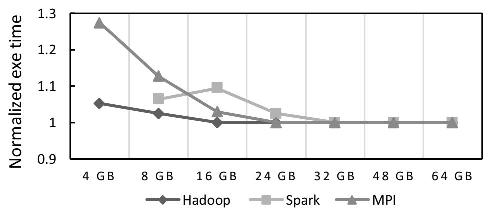

Figure 4: Impact of memory capacity per node on performance

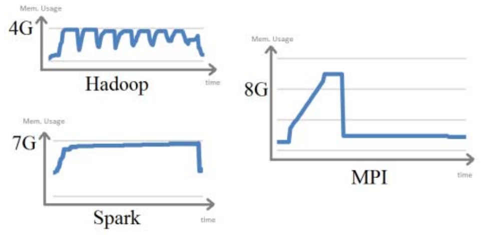

Figure 5: K-means memory usage on various frameworks

from Hadoop, wordcount, grep, PageRank, Bayes, and nBayes from Spark, and sort, BasicMPEG, and grep from MPI to be Disk-intensive while others to be CPU-intensive. This can be explained as follows: If increasing the processor's frequency reduces the active state residency (C0) of the processor, the application is I/O bound, as when a core is waiting for I/O, the core changes its state to save power. Similarly, if active state residency does not change the workload is CPU bound.

2) Disk implication on application behavior: To show how low speed disks can change the memory bandwidth usage of big data frameworks, we also performed same experiments using HDD. Figure 8 shows that changing the disk from HDD to SSD improves the performance of Hadoop, Spark, and MPI by 40%, 105%, and 183% respectively. The reason that MPI applications take more advantage from faster disk is that our studied MPI based applications are written in C++. While Hadoop and Spark are Java based frameworks and they use HDFS as an intermediate layer to access and manage storage. Consequently, MapReduce based frameworks are not as quick as MPI; hence faster storage or memory cannot bring a noticeable performance benefit for Hadoop and Spark compared to MPI framework.

Using one of the fastest SSD disk in the market did not help putting high pressure on memory subsystem (Our result has shown that the average memory bandwidth utilization is 18.8% when we use the minimum available memory bandwidth). Another point regarding the storage is to use multiple disks per node to alleviate IO bottleneck. We performed a new set of experiments with two SSD storages per node. We found that HDFS is not aware of multiple disks on the node and all data is written or read from one disk. Therefore, using multiple disks per node does not guarantee the parallel access to the data blocks of HDFS to reduce the IO bottleneck.

Also, we have increased aggregate storage by increasing the number of nodes (from 6 nodes to 12 nodes). We present the results in section 3.B.4. The result reveals that increasing the number of nodes not only reduces IO requests' pressure on each node, but also it reduces pressure on memory subsystem of each node. This is another reason that in real situation where there are thousands of nodes, emerging memory technologies do not bring noticeable performance benefit for studied big data applications. This is due to the fact that off-chip memory bandwidth demands of big data applications is low compared to the available bandwidth of current memory technologies.

It is important to note that SSD increases the read and write bandwidth of disk and substantially reduces the latency of access to disk compared to HDD. However, accessing to I/O means loosing of millions of CPU cycles, which is large enough to vanish any noticeable advantage of using a high-performance DRAM. Hence, a high bandwidth DRAM is not required to accelerate the performance of MapReduce frameworks in presence of a slow HDD or even a fast SSD.

3) Core count implication: In the previous section, we classified big data applications into two groups of CPU intensive and I/O intensive. Figure 9 demonstrates the effect of increasing the number of cores per node on the performance of these two groups. The expectation is that performance of the system improves linearly by adding cores because big data applications are heavily parallel. However, we observe a different trend. For CPU intensive applications and when the core count is less than 6 cores per node, the performance improvement is close to the ideal

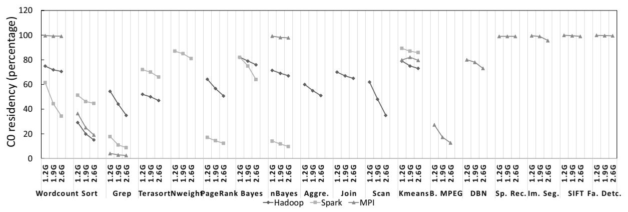

Figure 7: C0 residency of processor

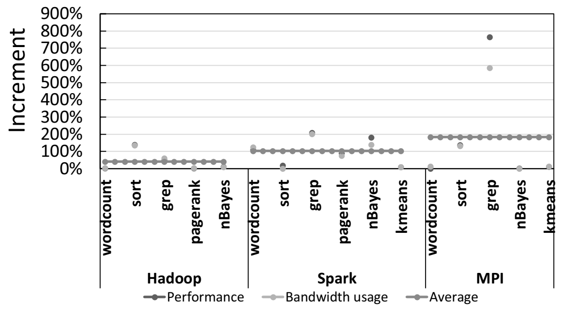

Figure 8: Effect of changing disk from HDD to SSD on execution time and memory bandwidth usage

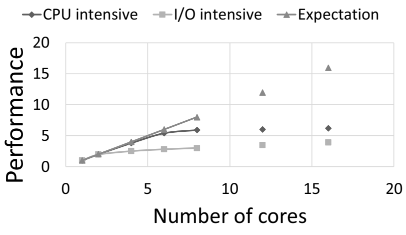

Figure 9: Effect of core count on performance

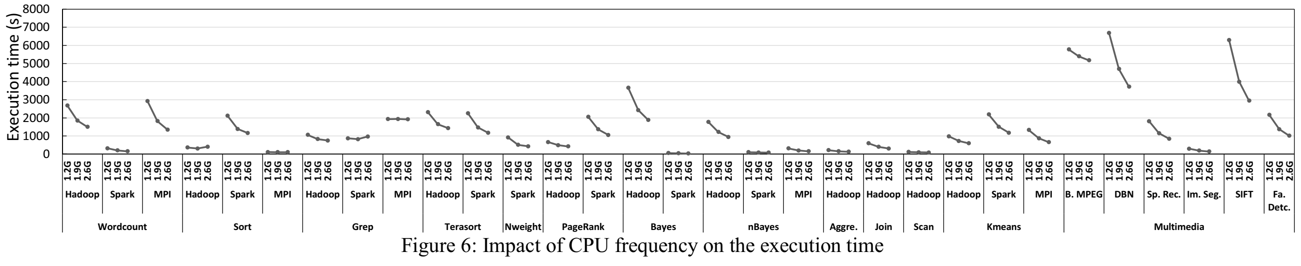

Figure 6: Impact of CPU frequency on the execution time

case. The interesting trend is that increasing the number of cores per node does not improve the performance of big data applications noticeably beyond 6 cores. As the increase in the number of cores increases the number of accesses to the disk, the disk becomes the bottleneck of the system. At 8 cores, the CPU utilization is dropped to 15% for I/O intensive applications, which proves that increasing the number of CPU's cores per node (scale-up approach) is not an effective solution to improve the performance of I/O intensive applications such as MapReduce frameworks.

It is important to note that our disk is SSD (Solid State Drive) and the performance benefit could have diminished at even lower number of cores if a HDD (Hard Disk Drive) was used. We also projected the performance of 12 and 16 cores by regression model derived from our experimental results which shows that increasing core count is not an effective solution to bring noticeable performance benefits for big data applications as it increases the Disk accesses and making it a bottleneck.

4) Cluster size implication: Our cluster size (6 nodes) is small compared to a real-world big data server cluster. It is therefore important to understand the impact of cluster size on the memory characterization results. To achieve this, we performed three additional experiments with a single node, a three-node, as well as a twelve-node cluster to study the effect of cluster size on memory subsystem performance. Figure 10 shows the result of our experiments. These results show that increasing the size of cluster from 1 to 12 changes the memory behavior; it slightly reduces the pressure on memory subsystem. Increasing the cluster size reduces both memory usage and memory bandwidth utilization on each node, on average. We anticipate that the memory subsystem mostly will not be under pressure in a large cluster.

5) Cache implication: Modern processor has a 3-level cache hierarchy. Figure 11 shows big data applications' cache hit ratio of level 2 (L2) and last level caches (LLC). The results reveal an important characteristic of big data applications. Contrary to the simulation's results in recent work reporting cache hit ratio to be below 10% for big data applications [3], our experimental result shows big data applications have a much higher cache hit ratio, which helps reducing the number of accesses to the main memory. The average cache hit ratio for big data applications implies that these applications are cache friendly. Consequently, CPUs with Larger LLC will show similar characteristic. Therefore, we can conclude that due to cache behavior, big data applications are not memory intensive unless they are iterative tasks.

This explains why increasing the memory frequency and increasing the number of channels does not improve the performance of most of big data applications except for iterative tasks such as K-means, Nweight, and Image Segmentation. The reason of high cache hit ratio is that each parallel task of big data frameworks processes data in a sequential manner. This behavior increases the cache hits; therefore it prevents excessive access to DRAM and eliminates the necessity of high bandwidth memory. This finding helps the server designers to avoid over provisioning the memory subsystem for big data applications.

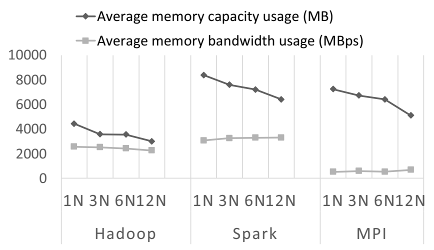

Figure 10: Impact of cluster size on memory capacity and bandwidth usage

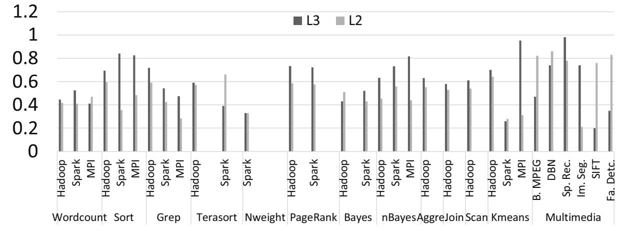

Figure 11: LLC and L2 hit rate

IV. REALATED WORK

A. Memory

A recent work on big data [3] profiles the memory access patterns of Hadoop and noSQL workloads by collecting memory DIMM traces using special hardware. A more recent work [4] provides a performance model that considers the impact of memory bandwidth and latency for big data, high performance, and enterprise workloads. The work in [5] shows how Hadoop workload demands different hardware resources. In [6] the authors evaluate contemporary multi-channel DDR SDRAM and Rambus DRAM systems in SMT architectures. The work in [11] mainly focuses on page table and virtual memory optimization of big data and [12] presents the characterization of cache hierarchy for a Hadoop cluster. Those works do not analyze the DRAM memory subsystem. The results of the latest work on memory characterization of Hadoop applications is in-line with our findings [28]. Moreover, [29] studied the impact of memory parameters on the power and energy efficiency of big data frameworks but did not study the effect of input size and processor configuration on memory behavior. Another recent work studied the effect of memory bandwidth on the performance of MapReduce frameworks and presented a memory navigator for modern hardware [30]. In addition, several studies have focused on memory system characterization of various non-big data workloads such as SPEC CPU or parallel benchmark suites [7, 8, 9, 10].

B. Big Data

A recent work on big data benchmarking [13] analyzes the redundancy among different big data benchmarks such as ICTBench, HiBench and traditional CPU workloads and introduces a new big data benchmark suite for spatio-temporal data. The work in [14] selects four big data workloads from the BigDataBench [21] to study I/O characteristics, such as disk read/write bandwidth, I/O devices utilization, average waiting time of I/O requests, and average size of I/O requests. Another work [15] studies the performance characterization of Hadoop and DataMPI, using Amdahl's second law. This study shows that a DataMPI is more balanced than a Hadoop system. In a more recent work [16] the authors analyze three SPEC CPU2006 benchmarks (libquantum, h264ref, and hmmer) to determine their potential as big data computation workloads. The work in [17] examines the performance characteristics of three high performance graph analytics. One of their findings is that graph workloads fail to fully utilize the platform's memory bandwidth. In a recent work [18], Principle Component Analysis is used to detect the most important characteristics of big data workloads from BigDataBench. To understand Spark's architectural and micro-architectural behaviors, a recent work evaluates the benchmark on a 17-node Xeon cluster [19]. Again, this study does not consider the effect of memory subsystems on big data. The work in [20] performs performance analysis and characterizations only for Hadoop K-means iterations.

V. Conclusion

Characterizing memory behavior of big data frameworks is important as it helps guiding scheduling decision in cloud scale architectures as well as helping making decisions in designing server cluster for big data computing. While latest works have performed a limited study on memory characterization of big data applications, this work performs a comprehensive analysis of memory requirements through an experimental evaluation setup. We study diverse domains of applications from microkernels, graph analytics, machine learning, E-commerce, social networks, search engines, and multimedia in Hadoop, Spark, and MPI. This gives us several insights into understanding the memory role for these important frameworks. We observe that most of studied big data applications in MapReduce based frameworks such as Hadoop and Spark do not require a high-end memory. On the other hand, MPI applications, as well as iterative tasks in Spark (e.g. machine learning) benefit from a high-end memory.

References

[1] Bertino et al., "big data-Opportunities and Challenges," in IEEE 37th Annual Computer Software and Applications Conf., pp. 479-480. 2013.

[2] The Apache Software Foundation, "What is Apache Hadoop?" Available at: https://hadoop.apache.org/

[3] M. Dimitrov et al., "Memory system characterization of big data workloads," in IEEE Conf. on big data, pp. 15-22, October 2013.

[4] R. Clapp et al., "Quantifying the Performance Impact of Memory Latency and Bandwidth for big data Workloads," in IEEE Symp. on Workload Characterization (*IISWC*) , pp. 213-224, October 2015.

[5] I. Alzuru et al., "Hadoop Characterization," in *Trustcom/BigDataSE/ISPA* 2015, Vol. 2, pp. 96-103, August 2015.

[6] Z. Zhu, and Z. Zhang, "A performance comparison of DRAM memory system optimizations for SMT processors," in the 11th *HPCA*, pp. 213-224, February 2005.

[7] Barroso et al., "Memory system characterization of commercial workloads," ACM SIGARCH Computer Architecture News 26, no. 3, pp. 3-14, 1998.

[8] A. Jaleel et al., "Memory characterization of workloads using instrumentation-driven simulation–a pin-based memory characterization of the SPEC CPU2000 and SPEC CPU2006 benchmark suites," Intel Corporation, VSSAD, 2007.

[9] F. Zeng et al., "Memory performance characterization of spec cpu2006 benchmarks using tsim," Physics Procedia 33, pp. 1029-1035, 2012.

[10] Shao et al., "ISA-independent workload characterization and its implications for specialized architectures," in *ISPASS*, pp. 245-255, 2013.

[11] Basu et al., "Efficient virtual memory for big memory servers," ACM SIGARCH Computer Architecture News, vol. 41, no. 3, pp. 237-248, 2013.

[12] Jia et al., "Characterizing data analysis workloads in data centers," in IEEE *IISWC*, pp. 66-76, 2013.

[13] W. Xiong et al., "A characterization of big data benchmarks," in IEEE Conf. on big data, pp. 118-125, October 2013.

[14] F. Pan et al., "I/O characterization of big data workloads in data centers," in *BPOE*, pp. 85-97, 2014.

[15] F. Liang et al., "Performance characterization of hadoop and data mpi based on amdahl's second law," in *NAS*, pp. 207-215, August 2014.

[16] K. Hurt, and E. John, "Analysis of Memory Sensitive SPEC CPU2006 Integer Benchmarks for big data Benchmarking," in Proc. *PABS*, pp. 11-16, February 2015

[17] S. Beamer et al., "Locality exists in graph processing: Workload characterization on an Ivy Bridge server," in the IEEE *IISWC*, pp. 56-65, October 2015.

[18] Z. Jia et al., "Characterizing and subsetting big data workloads," in IEEE *IISWC*, pp. 191-201, October 2014.

[19] T. Jiang et al., "Understanding the behavior of in-memory computing workloads," in the IEEE *IISWC*, pp. 22-30, 2014.

[20] Issa, J. "Performance characterization and analysis for Hadoop K-means iteration" Journal of Cloud Computing, 5(1), 1, 2015.

[21] Lei et al. "Bigdatabench: A big data benchmark suite from internet services," in IEEE 20th *HPCA*, pp. 488-499, 2014.

[22] Available at: https://software.intel.com/en-us/articles/intel-performance-counter-monitor

[23] S. Huang et al., "The hibench benchmark suite: Characterization of the mapreducebased data analysis," in IEEE *ICDE*, pp. 41–51, 2010.

[24] M. Malik et al., "Characterizing Hadoop applications on microservers for performance and energy efficiency optimizations," in *ISPASS*, pp. 153-154, 2016.

[25] Ch. Bienia et al., "The PARSEC benchmark suite: characterization and architectural implications," Proc. *PACT*, pp. 72-81, 2008.

[26] Ferdman, Michael, et al. "Clearing the clouds: a study of emerging scale-out workloads on modern hardware." *ACM SIGPLAN Notices*. Vol. 47. No. 4. ACM, 2012.

[27] Ousterhout, Kay, et al. "Making Sense of Performance in Data Analytics Frameworks." *NSDI*. Vol. 15. 2015.

[28] H. M. Makrani, et al. "Understanding the Role of Memory Subsystem on Performance and Energy-Efficiency of Hadoop Applications." Proc. International Green and Sustainable Computing Conference (IGSC), 2017.

[29] H.M. Makrani, and H. Homayoun, "Memory requirements of hadoop, spark, and MPI based big data applications on commodity server class architectures." Proc. IEEE IISWC, pp. 112-113, 2017.

[30] H.M. Makrani, and H. Homayoun, "MeNa: A Memory Navigator for Modern Hardware in a Scale-out Environment." Proc. IEEE IISWC, pp. 2-11, 2017.