A Comprehensive Memory Analysis of Data Intensive Workloads on Server Class Architecture

Hosein Mohammadi Makrani
George Mason University
Fairfax, Virginia
hmohamm8@gmu.edu

Hossein Sayadi
George Mason University
Fairfax, Virginia
hsayadi@gmu.edu

Sai Manoj Pudukotai Dinakarra
George Mason University
Fairfax, Virginia
saimanoj.p.2013@ieee.org

Setareh Rafatirad
George Mason University
Fairfax, Virginia
srafatir@gmu.edu

Houman Homayoun
George Mason University
Fairfax, Virginia
hhomayou@gmu.edu

ABSTRACT

The emergence of data analytics frameworks requires computational resources and memory subsystems that can naturally scale to manage massive amounts of diverse data. Given the large size and heterogeneity of the data, it is currently unclear whether data analytics frameworks will require high performance and large capacity memory to cope with this change and exactly what role main memory subsystems will play; particularly in terms of energy efficiency. In this paper, we investigate how the choice of DRAM (high-end vs low-end) impacts the performance of Hadoop, Spark, and MPI based Big Data workloads in the presence of different storage types on a local cluster. Our results show that Hadoop workloads do not require high capacity memory. However, Spark and MPI based workloads require large capacity memory. Moreover, Increasing memory bandwidth through the increasing memory frequency or the number of channels does not improve the performance of Hadoop workloads while iterative tasks in Spark and MPI benefits from high bandwidth memory. Among the configurable parameters, our results indicate that increasing the number of DRAM channels reduces DRAM power and improves the energy-efficiency across all applications.

CCS CONCEPTS

• **Hardware → Memory and dense storage**; **Memory and dense storage**; • **Computer systems organization** → *Multicore architectures*; *Cloud computing*; *Multicore architectures*; *Cloud computing*;

KEYWORDS

Memory, DRAM, Characterization, Performance, Power, Big Data, Hadoop, Spark, In-Memory processing

MEMSYS, October 1–4, 2018, Old Town Alexandria, VA, USA

ACM ISBN 978-1-4503-6475-1/18/10…$15.00
https://doi.org/10.1145/3240302.3240320

ACM Reference Format:
Hosein Mohammadi Makrani, Hossein Sayadi, Sai Manoj Pudukotai Dinakarra, Setareh Rafatirad, and Houman Homayoun. 2018. A Comprehensive Memory Analysis of Data Intensive Workloads on Server Class Architecture. In *The International Symposium on Memory Systems (MEMSYS), October 1–4, 2018, Old Town Alexandria, VA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3240302.3240320

1 INTRODUCTION

Big data is an enabler of future strategies and immediate change through the power of predictive analytics and advanced data science. Properly harnessing data can help to achieve better, fact-based decision-making and improve the overall customer experience. By using new big data technologies, companies can answer questions in seconds rather than days, and in days rather than months. This acceleration allows businesses to enable the type of quick reactions to key business questions and challenges that can build competitive advantage and improve performance, and provide answers for complex problems or questions that have resisted analysis.

Big data analytics applications heavily rely on machine learning and data mining algorithms [34, 35], and are running complex software stack with significant interaction with I/O and OS, and exhibit high computational intensity and I/O intensity [9]. 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.

Big data frameworks such as Hadoop, Spark, and MPI are three popular platform that enables big data analytics. Hadoop has been developed to use a cluster of commodity server to process large datasets. However, Spark is developed to overcome the limitation of Hadoop on efficiently utilizing main memory. MPI, a de facto industry standard for parallel programming on distributed memory systems, is also another platform used for data analytics [37].

In the era of big data, it is important to evaluate the effect of main memory parameters on the performance of data intensive applications in the presence of different storage types. While there is literature 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 [4, 12, 16, 17, 20, 27, 31].

In this paper, we evaluate the impact of the memory parameters on the performance and energy efficiency of big data analytics 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. Additionally, we study the impact of storage on the memory behavior 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 1866 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 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 not over-provision 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. To understand whether our observations on memory subsystem behavior remains valid for future architectures with higher number of cores, larger cache capacity, and higher operating frequency, we performed further micro-architectural study to understand the impact of these parameters on memory behavior. Our results suggest to use a low frequency DRAM memory with high number of channels which reduces the power consumption of DRAM by 57% without any performance degradation in order to improve the energy efficiency of big data clusters.

The findings of this study are important as they help server designers to avoid over provisioning the memory subsystem for many of data analytics applications. Moreover, we found that the current storage systems are the main bottleneck for the studied applications hence any further improvement of memory and CPU architecture without addressing the storage problem is a waste of money and energy.

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 describes related works. Finally, Section 5 concludes the paper.

2 EXPERIMENTAL SETUP

In this section, we present our experimental system configurations and its setup. We first introduce the studied frameworks and workloads. We then describe our hardware platform. Finally, we present experimental methodology and the tuning of HDFS (Hadoop Distributed File System) block size in order to optimize the platform for Hadoop and Spark frameworks.

2.1 Frameworks

Hadoop: One of the most popular framework for big data is MapReduce introduced by Google. Apache Hadoop is a java-based open source implementation of the MapReduce programming model, which is pivotal in big data computing [1]. Hadoop has been utilized in various areas, such as machine learning, search engines, log analysis, and e-commerce. The success of Hadoop is due to its scalability, fault tolerance, and simplicity of programming. Hadoop is composed of two layers. The first layer is a data storage called HDFS and the second layer is a data processor called Hadoop MapReduce framework. HDFS is a block based file system. Hadoop MapReduce cannot keep reused data and state information during execution [14]. Hence, it has to iteratively read the same data in each iteration, which results in significant disk accesses and unnecessary overhead.

Spark: Spark is another MapReduce-like cluster computing framework designed to overcome Hadoop's shortage in utilizing main memory. Spark uses HDFS as data storage system. In addition, Spark uses a new data structure called Resilient Distribute Dataset (RDD). The main responsibility of RDD is to cache data, which avoids data reloading from the disk. RDD allows users to cache the high value data in memory, and controls the persistence of data. It is suitable for applications with iterative algorithms that can achieve tremendous speed up. Moreover, Spark supports a Directed Acyclic Graph (DAG) schedule which avoids materializing the intermediate values by pipeline operations to decrease I/O accesses. While Hadoop uses a heartbeat scheduler to communicate scheduling decisions which impose 5 to 10 second delay, Spark task scheduling is low latency through an event-driven architecture.

MPI: A peer-to-Peer network is a decentralized and distributed network where thousands of machines connected in the network consume, as well as, serve resources. Nodes use Message Passing Interface (MPI) to communicate and exchange data between themselves. One of the features of MPI is that a process does not need to read same data over and over because it can live as long as the system runs. However, MPI has a major drawback of lacking fault tolerance. Each node in this network is a single point of failure that can cause the whole system to shut down. Hence, users who prefer a robust and fault tolerant framework exploit other big data framework such as Hadoop. In response to this issue, there are plans to include fault-tolerance inside the MPI model in its next major release. In this paper, our focus is not on MPI applications but on data analytics workloads which use MPI for parallel implementation. The nature of most of big data applications is simple but the main challenge is to process big amount of data which is out of the capability of a single server to process. Therefore, findings of this paper regarding MPI are only valid for studied applications.

It is important to note that, there are complex MPI based applications (outside of the scope of this study) that could have completely different behavior than what we have observed in our experiments.

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 16.04 LTS. Our JVM version is 1.8.

2.2 Workloads

Big data analytics applications are characterized by four critical features, referred as the four "Vs": volume, velocity, variety, and veracity. Big data is inherently large in volume. Velocity refers to how fast the data is coming in and to how fast it needs to be analyzed. Variety refers to the number and diversity of sources of data and databases, such as sensor data, social media, multimedia, text, and much more. Veracity refers to the level of trust, consistency, and completeness of data. The diversity of applications is important for characterizing big data frameworks. This diversity can enable users to optimize their programs considering the memory configuration of the framework.

Similarly, cluster designers can evaluate their candidate memory configurations by considering different classes of applications. Hence, 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 [37] and HiBench [14] for the choice of benchmarking. We selected a diverse set of applications and frameworks to be representative of data analytics domain. More details of these workloads are provided in Table 1. 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.

2.3 Hardware platform

We carefully selected our experimental platform to investigate the micro-architectural effect on the performance of data analytics frameworks to understand whether our observations on memory subsystem behavior remains valid for future architectures with enhanced microarchitecture parameters or not. This includes analyzing the results when increasing the core count 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 data intensive applications.

For running the workloads and monitoring statistics, we used a six-node standalone cluster with detailed characteristics presented in Table 2. To have a comprehensive experiment we used different SDRAM memory modules. All modules are provided from the same vendor. We used single socket servers in this study, in order to hide the NUMA effect (to understand DRAM-only impact). 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 [30], a modern high speed network introduces only a small 2% performance benefit. 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.

2.4 Methodology

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

Data collection: We used Intel Performance Counter Monitor tool (PCM) [2] 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 amount 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 [2], which provides the detailed power consumption of each socket and DRAM. We did not use WattsUp power meter because it does not have the breakdown of power and also it collects power consumption of several parts of the system which are not related to this study. Throughout this paper we will present the results based on high speed SSD disk. The default values for experiments are as follow: DRAM capacity=32GB, number of memory channels=4, memory frequency=2400 MHz, core count per CPU=16, CPU frequency=2.6 GHz.

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 [13], 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. Adjusting default memory parameters of Hadoop and Spark also is important. Hence, we tuned Hadoop and Spark memory related configuration parameters. Followings are two most important memory related parameters that we tuned for all experiments:

mapreduce.map.memory.mb: is the upper memory limit that Hadoop allows to be allocated to a mapper, in megabytes. *spark.executor.memory*: Amount of memory to use per executor process in Spark (e.g. 2g, 8g).

We set those values according to the following (we reserved 20% of DRAM capacity for OS):

$$mapreduce.map.memory.mb = (DRAMcapacity \times 0.8)/ Numberofconcurrentmapperspernode \quad (1)$$

$$spark.executor.memory = ((DRAMcapacity - spark.driver.memory) \times 0.8)/ Numberofexecutorpernode \quad (2)$$

Table 1: Studied workloads

Workload	Domain	Input type	Input size (huge)	Framework	Suite
Wordcount	micro-kernel	text	1.1 TB	Hadoop, Spark, MPI	BigData Bench
Sort	micro-kernel	data	178.8 GB		
Grep	micro-kernel	text	1.1 TB		
Terasort	micro-kernel	data	834 GB	Hadoop, Spark	
Naive Bayes	E-commerce	Data	306 GB	Hadoop, Spark, MPI	
Page Rank	E-commerce	Data	306 GB	Hadoop, Spark	
Bayes	E-commerce	Data	306 GB	Hadoop, Spark	HiBench
k-means	Machine learning	Graph	112.2 GB	Hadoop, Spark, MPI	BigDataBench
nweight	Graph analytics	Graph	176 GB	Spark	HiBench
Aggregation	Analytical query	Data	1.08 TB	Hadoop	
Join	Analytical query	Data	1.08 TB		
Scan	Analytical query	Data	1.08 TB		
B.MPEG	Multimedia	DVD stream	437 GB	MPI	BigDataBench
DBN	Multimedia	Images	MNIST Dataset		
Speech recognition	Multimedia	Audio	252 GB		
Image segmentation	Multimedia	Images	162 GB		
SIFT	Multimedia	Images	162 GB		
Face detection	Multimedia	Images	162 GB		

Table 2: Hardware Platform

Hardware type	Parameter	Value
CPU	Model	Intel Xeon E5-2683 V4
	# Core	16 (32 thread)
	Base Frequency	2.1 GHz
	Turbo Frequency	3.0 GHz
	TDP	120
	L3 Cache	40 MB
	Memory Type Support	DDR4 1866/2133/2400
	Maximum Memory Bandwidth	76.8 GB/S
	Max Memory Channels supported	4
Disk (SSD PCIE)	Model	Samsung 960 PRO M.2
	Capacity	512 GB
	Speed	Max 3.5 GB/S
Disk (SSD SATA)	Model	HyperX FURY
	Capacity	480 GB
	Speed	500 MB/S
Disk (HDD)	Model	Seagate
	Capacity	500 GB
	Speed	7200 RPM
Network Interface card	Model	ST1000SPEXD4
	Speed	1000 Mbps

A recent work has shown that among other tuning parameters in a MapReduce framework, HDFS block size is also influential on the performance [10]. HDFS Block size has a direct relation to the number of parallel tasks (in Spark and Hadoop), as shown in EQ. (3).

$$NumberofTasks = InputSize/BlockSize \quad (3)$$

In the above equation, the input size is the size of data that is distributed among nodes. The block size is the 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 per node and it has the highest frequency and number of channels. Table 3 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 3 configuration. We will present the classification of applications into CPU-intensive, I/O-intensive, and memory-intensive tasks in next section. Our tuning methodology helps to put high pressure on memory subsystem.

Table 3: HDFS block size tuning

Application class	*Input size*/(*# nodes* × *#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

3 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 data analytics frameworks and memory requirements. We also discuss the impact of storage system, and size of input data 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.

3.1 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. As the focus of this study is on the

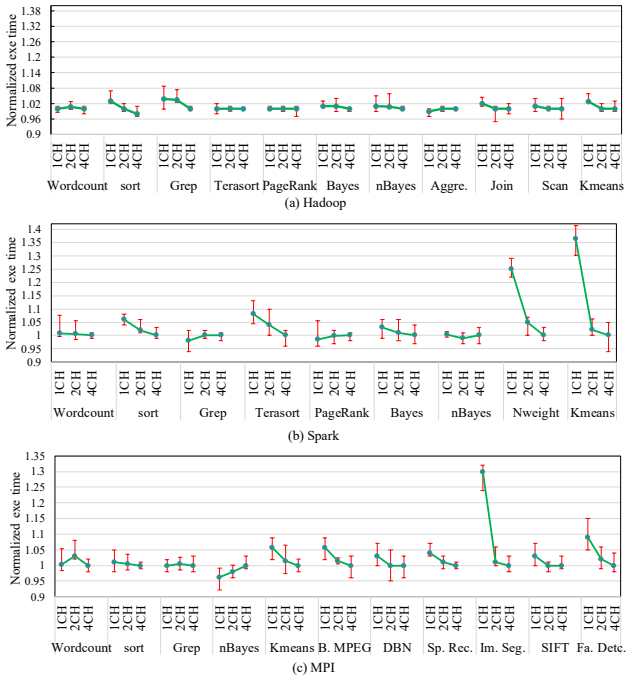

Figure 1: Effect of memory channel on the execution time (Normalized to 4CH)

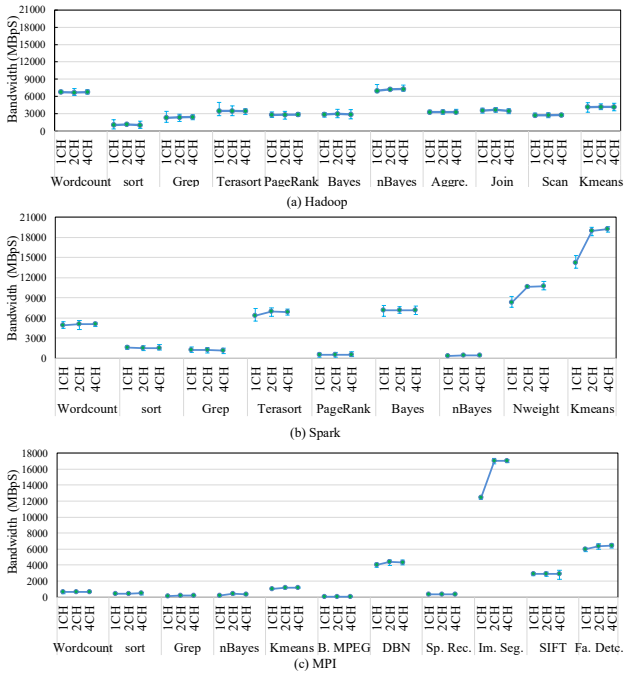

Figure 2: Impact of memory channel on bandwidth usage

memory subsystem, each memory related experiment has been performed 5 times. We present the average results, and also the minimum and maximum values of each set of experiments as an error bar.

Memory channels implication: The off-chip peak memory bandwidth equation is shown in EQ. (4).

$$Bandwidth = Channels \times Frequency \times Width \quad (4)$$

We observe in Figure 1 that increasing the number of channels from 1 to 4 does not significantly reduces the execution time of Hadoop applications (on average 6%). However, the execution time of K-means and Nweight in Spark framework, and Image segmentation with MPI implementation reduces more than 30%. It is noteworthy that aforementioned workloads 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 memory bandwidth usages are shown to be substantially higher than other workloads. One reason is those workloads are iterative and the second reason is the low cache hit rate of these application that cause excessive access to main memory (we will present that cache hit rate later in this paper). Therefore, providing more bandwidth improves their performance. By increasing the number of channels from 1 to 4, the performance gain is found to be 38%.

Memory frequency implication: As results in Figure 3 shows, similarly we don't observe significant reduction of execution time by increasing memory frequency (from 1866 MHz to 2400 MHz) for most of Hadoop applications. This finding may mislead to use the lowest memory frequency for Hadoop applications. Based on EQ. (5), read latency of DRAM depends on the memory frequency.

$$Readlatency = 2 \times (CL/Frequency) \quad (5)$$

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 Hadoop applications. In the other hand, we observe that iterative tasks in Spark and MPI require high frequency memory and their execution time reduces by high bandwidth memory.

DRAM capacity implication: To investigate the impact 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. An interesting observation is that a large memory capacity has not impact on the performance of studied Hadoop workloads. 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. In our experiments, the memory capacity usage of studied Hadoop applications never exceeded 4 GB on each node.

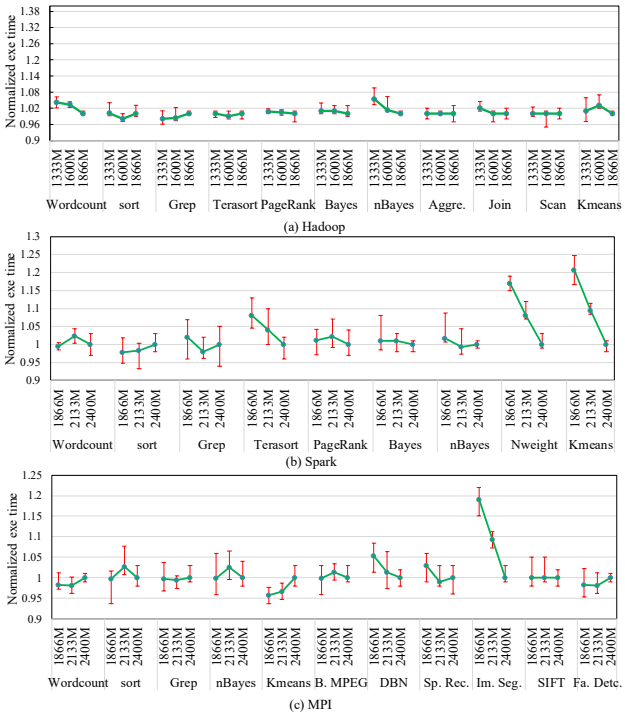

Figure 3: Effect of memory frequency on the execution time (Normalized to 2400MHz)

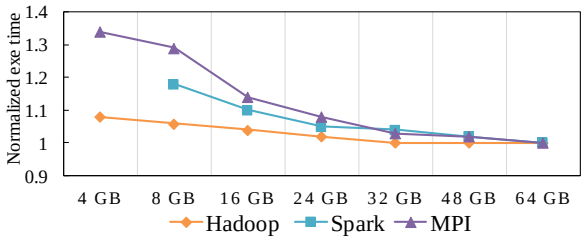

Figure 4: Impact of memory capacity per node on performance

However, Spark and MPI based applications show different behavior. 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 nodes 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 frameworks in Figure 4 (Normalized to 64 GB). To illustrate how these frameworks utilize DRAM capacity we present K-means memory usage on different frameworks in Figure 5.

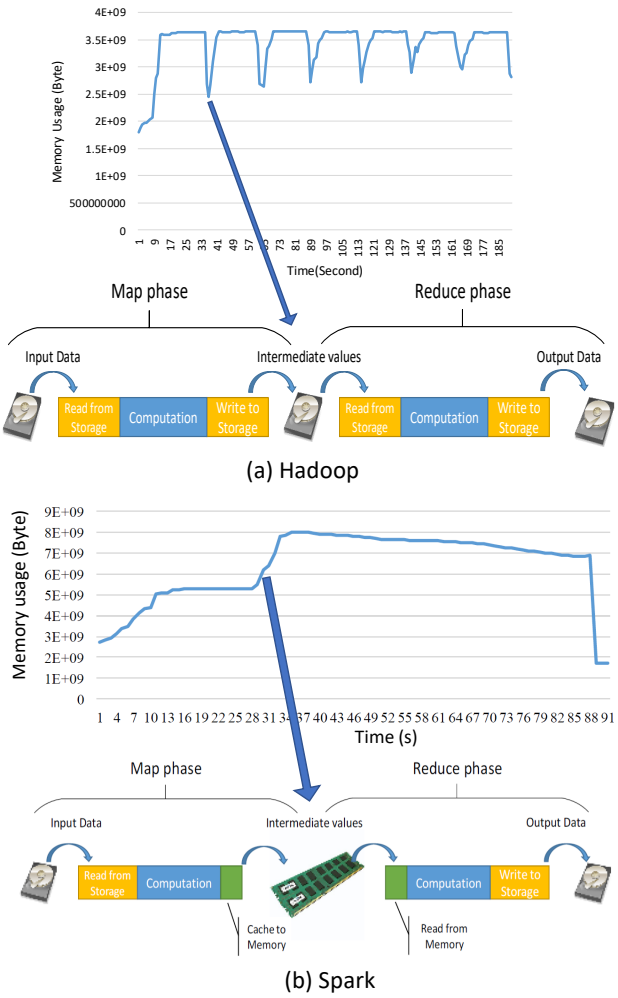

Figure 5: K-means memory usage on various frameworks

Input size implication: Today the paradigm has been shifted and new MapReduce processing frameworks such as Hadoop and Spark are emerging. Hadoop uses disk as storage and rely on a cluster of servers to process data in a distributed manner. The ability of hadoop 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 this framework. In the other hand, Spark is in-memory computing framework and changing input size have a large impact on the memory capacity usage. Fo MPI based applications, the extent of impact of input size on memory capacity usage depends on the application 's implementation. Although, for most of MPI based applications, we observed that memory capacity usage increases by increasing input size.

Another parameter that can be affected by the size of input data is the memory bandwidth usage. Our results reveal that the size of input data does not noticeably change the memory behavior of big

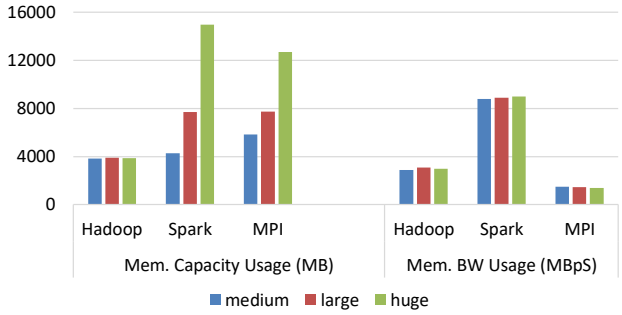

Figure 6: Average results of input size's effect on memory behavior

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. Figure 6 shows the average results of workloads from Hibench benchmark. We have performed experiments with 3 sets of input data namely medium, large, and huge (these are keywords of Hibench for input generation).

3.2 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. In this section, we present the classification of workloads into memory bound, compute bound, and I/O bound based on architectural behavior, which helps to accurately present the relation of performance and workload characteristics.

Classification of workloads: As the goal of this section is to study the combined impact of node architecture and data intensive workload's characteristics, it is important to classify those workloads. To this goal, we have explored the architectural behavior of studied workloads to classify them and find more insights.

1) Core frequency implication: Figure 7 shows that studied workloads behave in two distinct ways. The execution time of the first group is decreased linearly by increasing the core frequency. The second groupâĂŹs 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 studied workloads have distinct behaviors of being either CPU bound or I/O bound. This conclusion further advocated by C0 state residency of processor in Figure 8. This proves sort, grep, PageRank, and scan from Hadoop, wordcount, grep, PageRank, Bayes, and nBayes from Spark, and sort, BasicMPEG, and grep from MPI to be I/O bound while others to be CPU bound. This can be explained as follows: If increasing the processorâĂŹs frequency reduces the active state residency (C0) of the processor, the workload is I/O bound, as when a core is waiting for I/O, the core changes its state to save power.

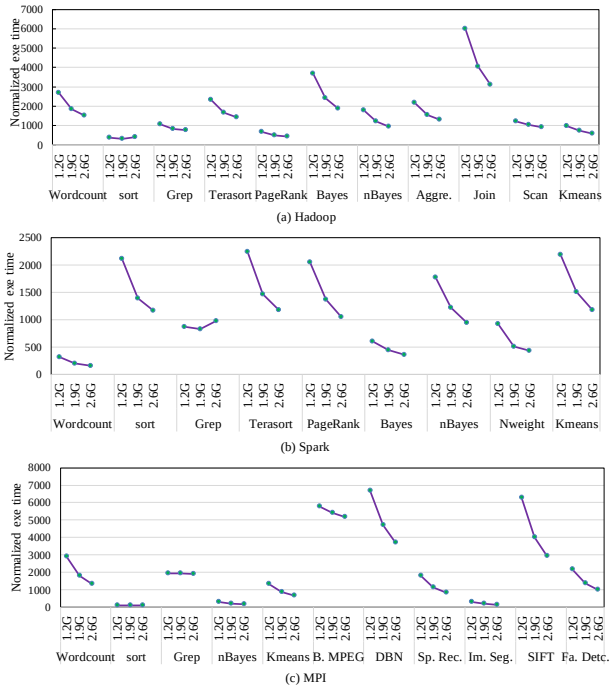

Figure 7: Impact of CPU frequency on the execution time

Similarly, if active state residency does not change the workload is CPU bound.

2) Cache implication: Modern processor has a 3-level cache hierarchy. Figure 9 shows cache hit ratio of level 2 (L2) and last level cache (LLC). The results reveal an important characteristic of data intensive workloads. Our experimental results show most of the studied workloads (particularly MapReduce workloads) have a much higher cache hit ratio, which helps reducing the number of accesses to the main memory. Based on simulation as well as real-system experiment results in recent works, it is reported that these applications' cache hit rate is too low (under 10%) [4, 12] for a system with 10 MB of LLC and for having LLC hit rate of 40%, the system should have around 100 MB of LLC. However, our real system experimental results show that most of data intensive workloads have much higher LLC hit rate (more than 50%) with only 40 MB LLC.

The reason of high cache hit ratio is that each parallel task of MapReduce framework processes data in a sequential manner. This behavior increases the cache hits; therefore it prevents excessive access to DRAM. Hence, based on the cache hit ratio of workloads and the intensity of accesses to memory, we can classify them into memory bound. If the cache hit ratio is low and the workload is an iterative task, it is classified as memory intensive. Our characterization showed that N-weight and Kmeans from Spark, and Image Segmentation from MPI are memory intensive. Therefore, we divided our workloads into three major groups of I/O bound, compute bound, and memory bound.

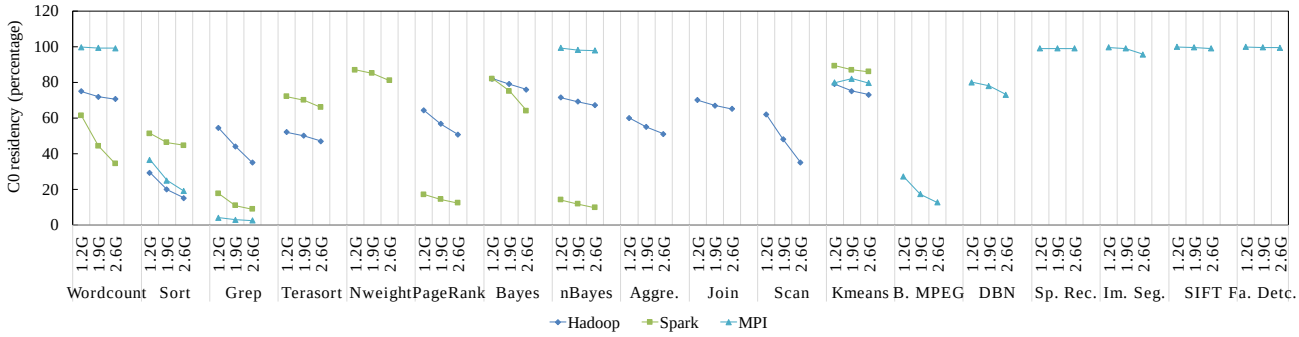

Figure 8: C0 residency of processor

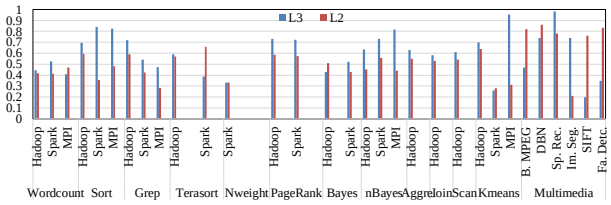

Figure 9: LLC and L2 hit rate

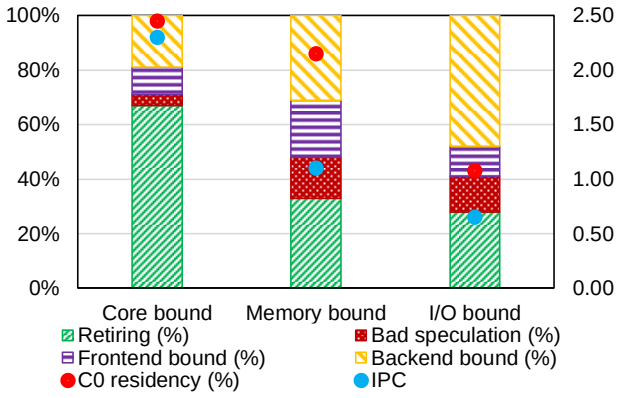

Figure 10: Average micro-architectural information of three classes

Top-Down methodology [40] chooses the micro-op (Âţop) queue of a out-of-order server as a dividing point between a coreâĂŹs front-end and back-end, and uses it to classify Âţop pipeline slots in four broad categories: Retiring, Front-end bound, Bad speculation, Back-end bound. Out of these, The Retiring classifies as âĂIJuseful workâĂİ and the rest prevent the workload from utilizing the full core width. We apply this approach to the our big data workloads and Figure 10 illustrates the micro architectural differences between three classes. Based on this classification, we present our result in the following sections.

Disk implication: To show how choice of storage can change the performance while using different memory configuration, we performed several experiments using three types of storage (HDD, SSD SATA, and SSD PCIe). Figure 11 shows that changing the disk from HDD to SSD PCIe improves the performance of Spark, Hadoop, and MPI by 1.6x, 2.4x, and 3.3x respectively. The reason that MPI workloads take more advantage from faster disk is that these workloads are written in C++. However, Hadoop and Spark are Java based frameworks and they use HDFS as an intermediate layer to access and manage storage. Our results show a high bandwidth DRAM is not required to accelerate the performance of MapReduce frameworks in presence of a slow HDD. However, MPI based workloads has the potential to benefit from high-end DRAM.

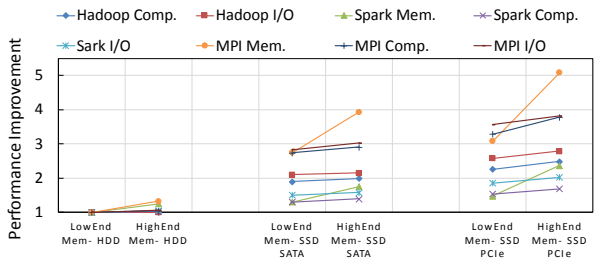

Figure 11: Effect of memory and storage configuration on the performance

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. While the performance will improve for IO intensive applications but using multiple disks per node does not guarantee the parallel access to the data blocks of HDFS to reduce the IO bottleneck. Another point is to use RAID. Since HDFS is taking care of fault-tolerance and "striped" reading, there is no need to use RAID underneath an HDFS. Using RAID will only be more expensive, offer less storage, and also be slower (depending on the concrete RAID configuration). Since the Namenode is a single-point-of-failure in HDFS, it requires a more reliable hardware setup. Therefore, the use of RAID is recommended only on the Namenodes.

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 losing of millions of CPU cycles, which is large enough to remove any noticeable advantage of using a high-performance DRAM. On the other hand, the only way to take advantage of a SSD is to read or write a big file (hundreds of megabyte) at once but our result shows that HDFS reads the data in much smaller blocks, regardless of HDFS block size. Figure 12 demonstrates the memory bandwidth utilization of each class on different storage type. Bandwidth utilization of memory bound workloads is shown to be substantially higher than other workloads and the results shows Using low speed disk reduces 55% the memory bandwidth usage and therefore prevents to get performance benefit from high bandwidth DRAM.

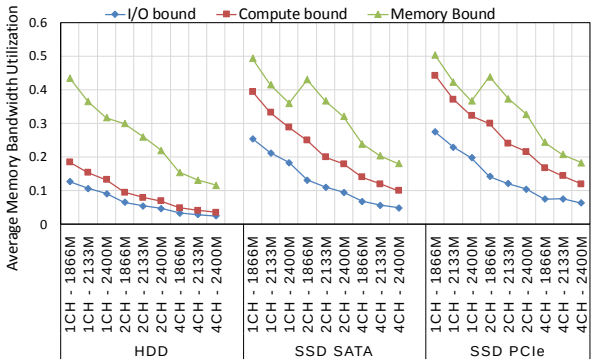

Figure 12: Average memory bandwidth utilization

Core count implication: In the previous section, we classified workloads into three groups. Figure 13 demonstrates the effect of increasing the number of cores per node on the performance of two groups of CPU intensive and IO intensive. The expectation is that performance of the system improves linearly by adding cores because big data workloads are heavily parallel. However, we observe a different trend. For CPU intensive workloads and when the core count is less than 6 cores per node, the performance improvement is close to the ideal case. The interesting trend is that increasing the number of cores per node does not improve the performance of data intensive workloads 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 44% for I/O intensive workloads, on average. This experiment performed when storage was SSD PCIe.

Based on those observations, we develop Eq. (6) to find the number of cores for which further increase does not noticeably enhance the performance of system:

$$Max(cores) = ((BW \times Nd))/((Nsc \times Fr \times \lambda)) \quad (6)$$

We define the variables used in this equation as follow: BW is the nominal bandwidth of each disk. Nd is the number of disk installed on the server. Nsc is the number of sockets. Fr is CPU core frequency

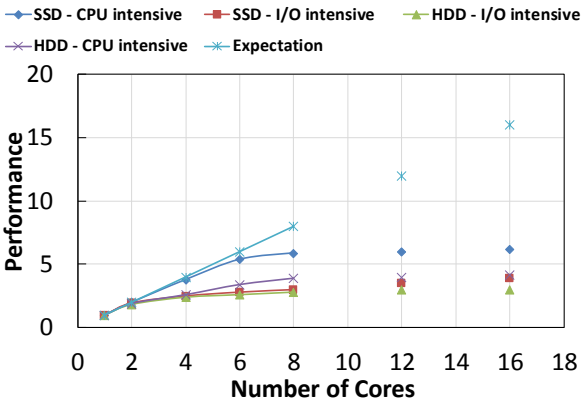

Figure 13: Effect of core count on the performance

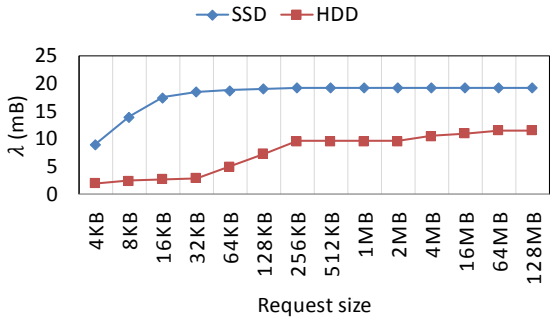

Figure 14: Lambda value for different request size and storage type

IO Request Size (IRS)	IRS = 16KB			32KB = IRS = 256KB			512KB = IRS = 128MB		
Workload class	Com.	Mem.	IO	Com.	Mem.	IO	Com.	Mem.	IO
Ave. error	5%	6%	15%	4%	4%	12%	4%	4%	7%

Figure 15: Average error of optimum core count prediction

and Lambda is a constant, which we found through our real-system experiments. As the effective I/O bandwidth depends on the block size and I/O request size, we have used fio [3] to calculate Lambda for different block size requests, presented in Figure 14. Designers can use this equation to select an optimum configuration, such as the number of cores, core frequency, disk type, and number of disks per node. As an example, the number of cores beyond which there is no noticeable performance gain on a server with one socket, one SSD storage with nominal 400 MBpS bandwidth, 16KB IO request size, running at 2.8 GHz is 8 based on the above equation. We have validated equation 1 for different classes of workload and the result is presented in Figure 15.

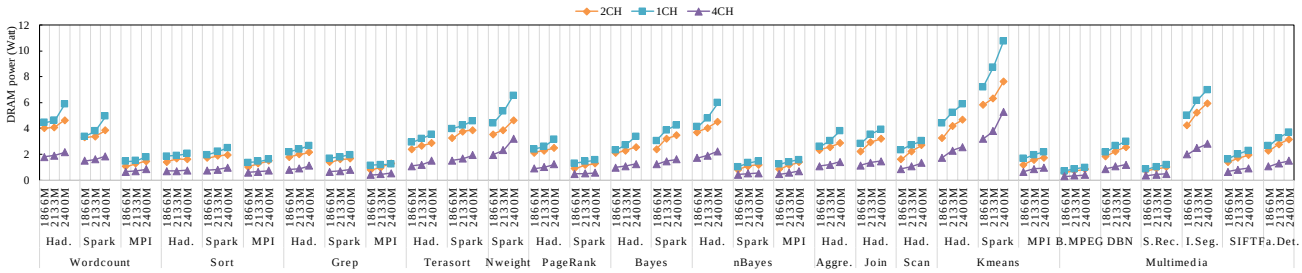

Figure 16: DRAM power consumption

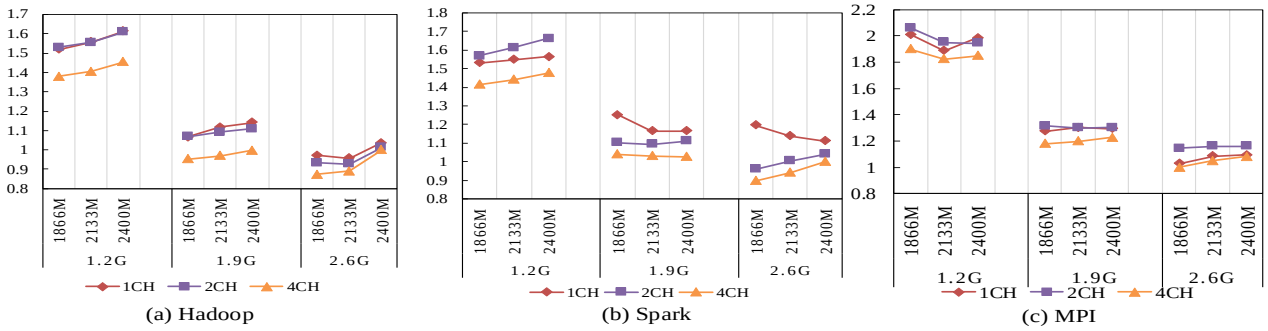

Figure 17: Average normalized EDP (Normalized to 4CH, 2400MHz, 2.6GHz)

3.3 Power Analysis

Figure 16 reports the DRAM power consumption. The first observation is that by increasing the frequency of DRAM by about 28% (1866 MHz to 2400 MHz), the power increases by almost 15%. Also, DRAM power increases when the core frequency raises as this increases the number of accesses to off-chip memory per unit of time. However, the DRAM power consumption is reduced when we increase the number of channels. An interesting observation is that a memory with 4 channels consumes 42% less power than a memory with 1 channel. This is due to the fact that DRAM channels are designed to increase the efficiency of interleaving. Thus, the memory controller can manage accesses more efficiently, which in turn reduces the power consumption.

In our experiments, the number of DIMMs/Channel is one. Despite the small/no impact on performance, increasing the number of memory channels to four significantly impacts the power consumption. Reducing the power of DRAM while increasing the number of channels can be explained as follow: Consider a 32 GB and 4 channel memory (occupied 4 DIMMs with 8 GB module), the memory controller does not need to put all modules in active state unlike the single channel. In single channel memory, one module with 32 GB capacity always must be in active state regardless of memory usage pattern. However, with 4 channels, memory controller can manage each channel individually and if there is no need to access to a channel, it can go to power saving mode. This increases the options for memory controller to perform power management.

Figure 17 depicts the average normalized Energy Delay Product (EDP) of frameworks. EQ. (7) indicates how we calculated this metric.

$$EDP = (CPUenergy + DRAMenergy) \times ExecutionTime \quad (7)$$

The results reveal that almost always increasing the number of channels improves EDP, as memory controller power management policy can benefit from such increase and reduce the power accordingly. Increasing the number of channels does not increase the execution time of studied applications. It also reduces DRAM power by up to 40%. Therefore, decreasing DRAM power decreases system power and consequently the energy of the system.

$$(Energy = Power \times Delay) \quad (8)$$

Hence, increasing the number of memory channels leads to an improvement in energy efficiency. The EDP's results of 1-channel and 2-channel memory are very similar. The trend in this figure shows that increasing the memory frequency increases EDP across all CPU operating frequency points. This implies that a high frequency off-chip memory is not an appropriate choice for EDP optimization for this class of big data applications. Furthermore, we observe that the EDP at 1.9 GHz CPU frequency is close to 2.6 GHZ. It shows that running the processor with highest frequency is not always necessary. A configuration with a single channel running at 2400 MHz memory frequency when CPU is running at 1.2 GHz is shown to have the worst EDP. On the other hand, a configuration with

4-channel and 1866 MHz memory frequency when CPU is running at 2.6 GHz is shown to have the best EDP for big data applications.

4 RELATED WORK

4.1 Memory

A recent work on big data [12] profiles the memory access patterns of Hadoop and noSQL workloads by collecting memory DIMM traces using special hardware. This study does not examine the effects of memory frequency and number of channels on the performance of the system. A more recent work [11] provides a performance model that considers the impact of memory bandwidth and latency for big data, high performance, and enterprise workloads. The work in [4] shows how Hadoop workload demands different hardware resources. This work also studies the memory capacity as a parameter that impacts the performance. However, as we showed in this work, their finding is in contrast with ours. In [43] the authors evaluate contemporary multi-channel DDR SDRAM and Rambus DRAM systems in SMT architectures. The work in [7] mainly focuses on page table and virtual memory optimization of big data and [17] presents the characterization of cache hierarchy for a Hadoop cluster. These works do not analyze the DRAM memory subsystem. In addition, several studies have focused on memory system characterization of various non-big data workloads such as SPEC CPU or parallel benchmark suites [5, 6, 36, 42]. Moreover, [21] 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 [22]. Few works [23, 33, 41] studied the impact of fault tolerant techniques on the performance and memory usage of embedded system. These works do not analyze the memory subsystem. There are also works on memory architecture such as [32, 39] but they did not analyzed the impact of memory parameters on big data applications.

4.2 Big Data

A recent work on big data benchmarking [38] 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 [31] selects four big data workloads from the BigDataBench [37] 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 [20] 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 [15] the authors analyze three SPEC CPU2006 benchmarks (libquantum, h264ref, and hmmer) to determine their potential as big data computation workloads. The work in [8] 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]. Their results show that Spark workloads have different behavior than Hadoop and HPC benchmarks. Again, this study does not consider the effect of memory subsystems on big data. The work in [16] performs performance analysis and characterizations for Hadoop K-means iterations. This study has also proposed a performance prediction model in order to estimates performance of Hadoop K-means iterations, without considering the memory requirements. The results of the latest works on memory characterization of Hadoop applications also are in-line with our findings [24, 26]. Moreover, there are studies on hardware acceleration of Hadoop applications that do not analyze the impact of memory and storage on the performance [28, 29]. Makrani et al. proposed compressive sensing based accelerator for multimedia big data application to reduce the I/O bottleneck for getting performance gain from high-end memory [25].

5 CONCLUSION

This paper answers the important questions of whether some of important data analytics frameworks such as Hadoop, Spark and MPI require high-capacity and high performance DRAM memory and what the role of memory for energy-efficient processing of data intensive applications is. Characterizing memory behavior of 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 data intensive 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.

The contribution of this paper is to give an insight on the role the memory subsystem plays in the overall performance of the servers when running data analytics frameworks. Our experimental results illustrate that data intensive workloads show three distinct behaviors (CPU-intensive, Disk-intensive, and memory-intensive). Based on the results presented in this paper, we observed that Hadoop framework is not memory intensive. This means Hadoop does not require high frequency, and large number of channels memory for higher performance. Our results show MPI and Spark based iterative tasks benefit from high memory frequency and large number of channels. Among the configurable parameters, our results indicate that increasing the number of DRAM channels reduces DRAM power and improves the energy-efficiency.

Moreover, our result shows that changing the disk from HDD to SSD improves the performance of Spark, Hadoop, and MPI by 1.6x, 2.4x, and 3.3x respectively. However, I/O bandwidth caps the performance benefit of multicore CPU. Therefore, we developed an experimental equation to help designers to find the number of cores for which further increase does not enhance system performance noticeably. Moreover, we found that the current storage systems are the main bottleneck for the studied applications hence any further

improvement of memory and CPU architecture without addressing the storage problem is a waste of money and energy.

REFERENCES

[1] [n. d.]. *The Apache Software Foundation, Available at: https://hadoop.apache.org/.*

[2] [n. d.]. *Available at: https://software.intel.com/en-us/articles/intel-performance-counter-monito.*

[3] Sanjay P Ahuja, Thomas F Furman, Kerwin E Roslie, and Jared T Wheeler. 2013. Empirical Performance Assessment of Public Clouds Using System Level Benchmarks. *International Journal of Cloud Applications and Computing (IJCAC)* 3, 4 (2013), 81–91.

[4] I. Alzuru and et. al. 2015. Hadoop Characterization. In *IEEE Trustcom/BigDataSE/ISPA*.

[5] L. A. Barroso, K. Gharachorloo, and E. Bugnion. 1998. Memory system characterization of commercial workloads. In *Int. Symp. on Computer Architecture*.

[6] L. A. Barroso, K. Gharachorloo, and E. Bugnion. 1998. Memory system characterization of commercial workloads. In *Int. Symp. on Computer Architecture*.

[7] Arkaprava Basu, Jayneel Gandhi, Jichuan Chang, Mark D Hill, and Michael M Swift. 2013. Efficient virtual memory for big memory servers. In *ACM SIGARCH Computer Architecture News*, Vol. 41. ACM, 237–248.

[8] Scott Beamer, Krste Asanovic, and David Patterson. 2015. Locality Exists in Graph Processing: Workload Characterization on an Ivy Bridge Server. In *IEEE Int. Symp. on Workload Characterization*.

[9] Bertino and et. al. 2013. Big-data opportunities and challenges. In *Annual Computer Software and Applications Conf.*

[10] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. 2008. The PARSEC benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*. ACM, 72–81.

[11] R. Clapp and et. al. 2015. Quantifying the Performance Impact of Memory Latency and Bandwidth for Big Data Workloads. In *IEEE Int. Symp. on Workload Characterization*.

[12] M. Dimitrov and et. al. 2013. Memory system characterization of big data workloads. In *IEEE Int. Conf. on Big Data*.

[13] 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.

[14] Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang. 2010. The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. In *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*. IEEE, 41–51.

[15] Kathlene Hurt and Eugene John. 2015. Analysis of Memory Sensitive SPEC CPU2006 Integer Benchmarks for Big Data Benchmarking. In *Proceedings of the 1st Workshop on Performance Analysis of Big Data Systems*. ACM, 11–16.

[16] Joseph Issa. 2016. Performance characterization and analysis for Hadoop K-means iteration. *Journal of Cloud Computing* 5, 1 (2016), 3.

[17] Zhen Jia, Lei Wang, Jianfeng Zhan, Lixin Zhang, and Chunjie Luo. 2013. Characterizing data analysis workloads in data centers. In *Workload Characterization (IISWC), 2013 IEEE International Symposium on*. IEEE, 66–76.

[18] Zhen Jia, Jianfeng Zhan, Lei Wang, Rui Han, Sally A McKee, Qiang Yang, Chunjie Luo, and Jingwei Li. 2014. Characterizing and subsetting big data workloads. In *Workload Characterization (IISWC), 2014 IEEE International Symposium on*. IEEE, 191–201.

[19] Tao Jiang, Qianlong Zhang, Rui Hou, Lin Chai, Sally A Mckee, Zhen Jia, and Ninghui Sun. 2014. Understanding the behavior of in-memory computing workloads. In *Workload Characterization (IISWC), 2014 IEEE International Symposium on*. IEEE, 22–30.

[20] Fan Liang, Chen Feng, Xiaoyi Lu, and Zhiwei Xu. 2014. Performance characterization of hadoop and data mpi based on amdahl's second law. In *Networking, Architecture, and Storage (NAS), 2014 9th IEEE International Conference on*. IEEE, 207–215.

[21] Hosein Mohammadi Makrani and Houman Homayoun. 2017. Memory requirements of hadoop, spark, and MPI based big data applications on commodity server class architectures. In *Workload Characterization (IISWC), 2017 IEEE International Symposium on*. IEEE, 112–113.

[22] Hosein Mohammadi Makrani and Houman Homayoun. 2017. MeNa: A Memory Navigator for Modern Hardware in a Scale-out Environment. In *2017 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2–11.

[23] Hosein Mohammadi Makrani, Amir Mahdi Hosseini Monazzah, Hamed Farbeh, and Seyed Ghassem Miremadi. 2014. Evaluation of software-based fault-tolerant techniques on embedded OSâĂŹs components. In *International Conference on Dependability (DEPEND)*. 51–57.

[24] Hosein Mohammadi Makrani, Setareh Rafatirad, Amir Houmansadr, and Houman Homayoun. 2018. Main-Memory Requirements of Big Data Applications on Commodity Server Platform. In *18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE.

[25] Hosein Mohammadi Makrani, Hossein Sayadi, Sai Manoj Pudukotai Dinakarra, Setareh Rafatirad, and Houman Homayoun. 2018. Compressive Sensing on Storage Data: An Effective Solution to Alleviate I/O Bottleneck in Data-Intensive Workloads. In *29th Annual IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*.

[26] Hosein Mohammadi Makrani, Shahab Tabatabaei, Setareh Rafatirad, and Houman Homayoun. 2017. Understanding the role of memory subsystem on performance and energy-efficiency of Hadoop applications. In *Green and Sustainable Computing Conference (IGSC), 2017 Eighth International*. IEEE, 1–6.

[27] Maria Malik, Avesta Sasan, Rajiv Joshi, Setareh Rafatirah, and Houman Homayoun. 2016. Characterizing Hadoop applications on microservers for performance and energy efficiency optimizations. In *Performance Analysis of Systems and Software (ISPASS), 2016 IEEE International Symposium on*. IEEE, 153–154.

[28] Katayoun Neshatpour, Hosein Mohammadi Makrani, Avesta Sasan, Hassan Ghasemzadeh, Setareh Rafatirad, and Houman Homayoun. 2018. Design Space Exploration for Hardware Acceleration of Machine Learning Applications in MapReduce. In *The 26th IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*.

[29] Katayoun Neshatpour, Hosein Mohammadi Makrani, Avesta Sasan, Hassan Ghasemzadeh, Setareh Rafatirad, and Houman Homayoun. 2018. Architectural considerations for FPGA acceleration of Machine Learning Applications in MapReduce. In *International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation(SAMOS)*.

[30] Kay Ousterhout, Ryan Rasti, Sylvia Ratnasamy, Scott Shenker, Byung-Gon Chun, and V ICSI. 2015. Making Sense of Performance in Data Analytics Frameworks.. In *NSDI*, Vol. 15. 293–307.

[31] Fengfeng Pan, Yinliang Yue, Jin Xiong, and Daxiang Hao. 2014. I/O characterization of big data workloads in data centers. In *Workshop on Big Data Benchmarks, Performance Optimization, and Emerging Hardware*. Springer, 85–97.

[32] Sai Manoj PD, Jie Lin, Shikai Zhu, Yingying Yin, Xu Liu, Xiwei Huang, Chongshen Song, Wenqi Zhang, Mei Yan, Zhiyi Yu, et al. 2017. A scalable network-on-chip microprocessor with 2.5 d integrated memory and accelerator. *IEEE Transactions on Circuits and Systems I: Regular Papers* 64, 6 (2017), 1432–1443.

[33] Hossein Sayadi, Hamed Farbeh, Amir Mahdi Hosseini Monazzah, and Seyed Ghassem Miremadi. 2014. A data recomputation approach for reliability improvement of scratchpad memory in embedded systems. In *Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2014 IEEE International Symposium on*. IEEE, 228–233.

[34] Hossein Sayadi, H.M Makrani, Onkar Randive, Sai Manoj P D, Setareh Rafatirad, and Houman Homayoun. 2018. Customized Machine Learning-Based Hardware-Assisted Malware Detection in Embedded Devices. *The 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications (IEEE TrustCom-18)* (2018).

[35] Hossein Sayadi, Nisarg Patel, Sai Manoj P D, Avesta Sasan, Setareh Rafatirad, and Houman Homayoun. 2018. Ensemble Learning for Effective Run-Time Hardware-Based Malware Detection: A Comprehensive Analysis and Classification. *ACM/IEEE Design Automation Conference, DAC* (2018).

[36] Yakun Sophia Shao and David Brooks. 2013. ISA-independent workload characterization and its implications for specialized architectures. In *Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on*. IEEE, 245–255.

[37] Lei Wang, Jianfeng Zhan, Chunjie Luo, Yuqing Zhu, Qiang Yang, Yongqiang He, Wanling Gao, Zhen Jia, Yingjie Shi, Shujie Zhang, et al. 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.

[38] Wen Xiong, Zhibin Yu, Zhendong Bei, Juanjuan Zhao, Fan Zhang, Yubin Zou, Xue Bai, Ye Li, and Chengzhong Xu. 2013. A characterization of big data benchmarks. In *Big Data, 2013 IEEE International Conference on*. IEEE, 118–125.

[39] Dongjun Xu, Ningmei Yu, PD Sai Manoj, Kanwen Wang, Hao Yu, and Mingbin Yu. 2015. A 2.5-D memory-logic integration with data-pattern-aware memory controller. *IEEE Design & Test* 32, 4 (2015), 1–10.

[40] Ahmad Yasin. 2014. A top-down method for performance analysis and counters architecture. In *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*. IEEE, 35–44.

[41] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2–2.

[42] Fucen Zeng, Lin Qiao, Mingliang Liu, and Zhizhong Tang. 2012. Memory performance characterization of spec cpu2006 benchmarks using tsim. *Physics Procedia* 33 (2012), 1029–1035.

[43] Zhichun Zhu and Zhao Zhang. 2005. A performance comparison of DRAM memory system optimizations for SMT processors. In *Int. Symp. on High-Performance Computer Architecture*.