MeNa: A Memory Navigator for Modern Hardware in a Scale-out Environment

Hosein Mohammadi Makrani, Houman Homayoun
Electrical and computer engineering department
George Mason University
Fairfax, USA
{hmohamm8, hhmoayou}@gmu.edu

Abstract*—Scale-out infrastructure such as Cloud is built upon a large network of multi-core processors. Performance, power consumption, and capital cost of such infrastructure depend on the overall system configuration including number of processing cores, core frequency, memory hierarchy and capacity, number of memory channels, and memory data rate. Among these parameters, memory subsystem is known to be one of the performance bottlenecks, contributing significantly to the overall capital and operational cost of the server. Also, given the rise of Big Data and analytics applications, this could potentially pose an even bigger challenge to the performance of cloud applications and cost of cloud infrastructure. Hence it is important to understand the role of memory subsystem in cloud infrastructure and in particular for this emerging class of applications. Despite the increasing interest in recent years, little work has been done in understanding memory requirements trends and developing accurate and effective models to predict performance and cost of memory subsystem. Currently there is no well-defined methodology for selecting a memory configuration that reduces execution time and power consumption by considering the capital and operational cost of cloud. In this paper, through a comprehensive real-system empirical analysis of performance, we address these challenges by first characterizing diverse types of scale-out applications across a wide range of memory configuration parameters. The characterization helps to accurately capture applications' behavior and derive a model to predict their performance. Based on the developed predictive model, we propose MeNa, which is a methodology to maximize the performance/cost ratio of scale-out applications running in cloud environment. MeNa navigates memory and processor parameters to find the system configuration for a given application and a given budget, to maximum performance. Compared to brute force method, MeNa achieves more than 90% accuracy for identifying the right configuration parameters to maximize performance/cost ratio. Moreover, we show how MeNa can be effectively leveraged for server designers to find architectural insights or subscribers to allocate just enough budget to maximize performance of their applications in cloud.

Keywords—memory, performance modeling, cost optimization, cloud

I. INTRODUCTION

Cloud computing technology offers significant economic as well as social benefits [1]. Today, many enterprises are adopting cloud to reduce their capital and operational cost while meeting Quality of Service (QoS) goals [2]. At the same time, cloud subscribers expect to gain the maximum performance from the cloud resources, at the lowest cost. A significant portion of cloud infrastructure's system capital as well as operational cost is directly related to memory subsystem parameters [3], which also impacts performance of subscribers' application [4, 5, 6, 7].

Today, more applications are moving to the cloud. Therefore, for cloud-scale servers, the increasing number of cores and applications sharing off-chip memory makes its bandwidth as well as capacity a critical shared resource. These trends suggest that it is important to understand the role of memory parameters such as capacity, number of channels, and operating frequency on performance of emerging class of applications in scale-out environment. The main contribution of this study is setting out a roadmap for memory configuration to maximize the performance cost ratio of cloud infrastructure.

To the best of our knowledge there is no experimental work in understanding the impact of various memory parameters on the performance of emerging scale-out applications. An empirical evaluation is important as it provides the community with reliable and accurate outcomes, which can be used to identify trends and guide optimization decisions.

To this goal, we first analyze various applications architectural characteristics. Based on the characterization results we classify applications into four different classes namely CPU intensive, IO intensive, Hybrid Memory-CPU intensive, and Hybrid Memory-IO intensive. Based on this information we build a database and use it to drive an empirical performance model for each application class. Furthermore, we utilize IBM/SoftLayer TCO (total cost of ownership) calculator to drive a cost model for server platform in a scale-out environment such as cloud. The developed cost model takes into account the processor as well as memory parameters.

Based on the proposed predictive model, we present a novel methodology for selecting main memory parameters to maximize the performance per cost ratio of a given application in cloud. As the performance of memory subsystem depends on processor configuration, our methodology also navigates processor parameters as well as memory parameters (MeNa). MeNa is a three-stage methodology. It utilizes a fully connected Neural Network to classify a given application. After the classification, in the second stage, MeNa calculates the performance-cost sensitivity of application with respect to the server's parameters. In the third stage, MeNa solves a bounded knapsack problem using dynamic programming to

Table 1. Big Data Workloads

Workload	wordcount	sort	grep	pagerank	naïve bayes	kmeans
Domain	micro kernel	micro kernel	micro kernel	websearch	e-commerce	machine learning
Input type	text	data	text	data	data	graph
Input size	1.1 T	178.8G	1.1 T	16.8G	30.6G	112.2G
Framework	Hadoop, Spark	Hadoop, Spark	Hadoop, Spark	Hadoop, Spark	Hadoop, Spark	Hadoop, Spark
Suite	BigDataBench	BigDataBench	BigDataBench	BigDataBench	BigDataBench	BigDataBench

Table 2. Memory modules' part numbers

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

Table 3. IBM\SoftLayer bare metal servers

Processor type	#Socket	#Core	Core_freq	DRAM capacity	Disk bays	Net speed	Monthly charge
Xeon E3-1270	1	4	3.40 GHz	2 GB	2	2 Gbps	137 $
Xeon E5-2620	2	6	2.00 GHz	16 GB	12	10 Gbps	470 $
Xeon E5-2690	2	8	2.90 GHz	16 GB	12	10 Gbps	640 $
Xeon E7-4850	4	10	2.00 GHz	64 GB	6	10 Gbps	1602 $
Xeon E7-4890	4	15	2.80 GHz	128 GB	24	10 Gbps	2566 $

find a configuration, which maximizes the performance per cost ratio.

Utilizing MeNa and based on the characterization results we make the following major observations:

1) Hybrid Memory-CPU intensive applications performance benefit noticeably from increasing the number of cores, low frequency core, low frequency memory, and large number of memory channels. 2) IO intensive applications are benefiting from small number of cores, high frequency cores, low frequency memory, and small number of memory channels. 3) Despite diverse range of frequency available in the memory market, increasing the memory frequency does not show to improve performance/cost ratio. 4) Increasing the number of memory channels improves the performance/cost ratio of hybrid Memory-CPU intensive applications. 5) Increasing the number of sockets increases the performance/cost of the system only if the number of cores per socket increases accordingly. 6) Increasing the capital cost of a server or a target budget set by a user does not always enhance in the performance/cost ratio of applications.

The remainder of this paper is organized as follows: Section 2 provides technical overview of the investigated applications and the experimental setup. Characterization and results are presented in Section 3. Section 4 presents our performance and cost analysis. We propose our memory navigator model (MeNa) in section 5. Section 6 presents related works. Finally, section 7 concludes the work.

II. Experimental Setup

In this section, we present our experimental methodology and setup. We first present the studied applications and then introduce the studied big data software stacks. We will then describe our hardware platform and our experimental methodology.

A. *Workloads*

Diversity of applications is important for characterizing cloud platforms. Hence, we target three domains of applications from Big Data, multi-threaded programs, and CPU applications. For CPU and multithreaded applications we use SPEC CPU2006 [9] and PARSEC [10] benchmark suites, respectively. The studied big data applications are selected from BigDataBench suite [8], presented in table 1. BigDataBench has micro kernel applications as well as graph analytics and machine learning applications.

Table 4. Hardware Platform

Hardware Type	Parameter	Value
Motherboard	Model	Intel S2600CP2
CPU	Model	Intel Xeon E5-2650 v2
	# Core	8
	# Threads	16
	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
Network Interface Card	Model	ST1000SPEXD4
	Speed	1000 Mbps

B. *Hardware Platform*

To have a comprehensive analysis of memory subsystem we used different SDRAM modules shown in table 2. All modules are from the same vendor. To build a cost model, we used IBM SoftLayer TCO Calculator, based on datacenter SJC01 (Located in San Jose, CA). A list of some of available processor types is presented in Table 3. For running the workloads, and monitoring the main memory, CPU, and disk behavior, we used a six-node server with detailed parameters for each node presented in table 4.

Architectural Behavior. We used Intel Performance Counter Monitor tool (PCM) [11] to understand memory and processor behavior. The performance counter data are collected for the entire run of each application. We collect OS-level performance information with DSTAT tool—a profiling tool for Linux based systems. Some of the metrics that we used for study are memory footprint, memory bandwidth, L2, and Last Level Cache (LLC) hits ratio, instruction per cycle (IPC), and core C0 state residency.

III. Characterization and Results

In a cloud platform, architecture and configuration of the server directly impacts its TCO and performance. The extent of this impact depends on the sensitivity of a cloud application to the architectural parameters and system configurations. Hence, we need to evaluate the performance sensitivity of our workloads to those parameters. Based on the level of sensitivity, we will classify the studied workloads. We then explore the relation between performance and TCO, and architectural configurations for each application class. This approach helps to formulize the relationship among configuration of cloud's platform, performance, and cost.

A. Memory Analysis

We use IPC as a measure of application's performance. We consider the variation of workload's IPC, when we navigate memory and processor parameters, as an indicator for sensitivity of the application performance to those parameters.

1) Memory Sensitivity: Equation 1 expresses the memory bandwidth of the system as a function of number of channels, operating frequency and width.

$$Bandwidth = Channels \times Frequency \times Width \qquad \text{(Eq. 1)}$$

According to this equation, the maximum bandwidth that our platform supports is 59.7 GB/s (4 channel * 1.867 GHz * 8 Byte). Memory frequency is a characteristic of memory module and channel is the configuration of memory modules on the platform. The ability of using multiple channels effectively is decided by the support of memory controller. Because the focus of our study is on memory subsystem and its configuration, it is important to evaluate the sensitivity of our studied workloads to those parameters that are configurable, namely memory frequency, channels, and capacity. For our experiments, we used 3 sets of memory modules with different frequencies. A total of 22 different memory modules with a wide range of operating frequency, number of channels and capacity were selected based on their availability in the market for server class architectures. The memory modules frequency varies from 1333 MHz to 1867 MHz, number of channels ranges from 1 to 4, and their capacity is swept from 4 GB to 32 GB.

Table 5 (a) and (b) show IPC variation when increasing memory frequency and memory channel, respectively, for a subset of studied applications. The interesting observation is that Spark-Sort workloads is not sensitive to memory frequency. However it's the most sensitive application to the number of channels. Another interesting observation is that the sensitivity of most applications to memory channel is more than their sensitivity to memory frequency. Due to in-memory nature of Spark framework, we expected Spark applications to be more sensitive to memory frequency and the number of available channels compared to Hadoop applications. However, unexpectedly Hadoop applications are shown to be more sensitive.

2) Bandwidth Sensitivity: Based on Equation 1 and the parameters of the studied memory modules reported in table 4, the minimum bandwidth that studied memory modules supports is 10.6 GB/s and the maximum bandwidth is 59.7 GB/s. Given that the studied workloads have different memory behavior and requirements, for off-chip memory bandwidth study we classify applications into memory intensive and non-intensive applications. The classification is done based on IPC variation as a function of memory bandwidth reported earlier in this section. Figure 1 (a) presents the average utilization of off-chip bandwidth for each class of applications. According to this observation, memory intensive workloads use almost 4x more bandwidth than non-intensive workloads. This figure also shows that both memory intensive and non-intensive workloads cannot fully utilize the maximum available bandwidth. This implies the inefficiency of the modern server platforms when utilizing memory bandwidth. Our observation shows available memory bandwidth exceeds the needs of all studied applications from various domains by approximately 10x and off-chip bandwidth is not a bottleneck for increasing the number of cores.

Figure 1 (b) demonstrates the impact of core frequency on the average bandwidth usage of memory intensive workloads for two different memory configurations, one with maximum and the other with minimum memory bandwidth. The first configuration is a memory with one channel and memory frequency of 1333 MHz and the second is a four-channel memory and 1867 MHz frequency. Based on this figure, we observe that when the core frequency is low we can see both configurations can deliver required bandwidth for the workloads. However, by increasing the frequency, the bandwidth utilization of workloads is increasing. This is due to the fact that increasing processor frequency increases the number of memory request generates per unit of time. The bandwidth utilization gap between the two memory configurations increases when increasing the processor frequency. To show the impact of bandwidth utilization on performance, in Figure 1 (c) we report the speedup in terms of relative execution time improvement comparing the two memory configurations. Increasing the core frequency up to 1.8 GHz does not bring performance advantage when using a higher bandwidth memory. However, it is only at frequency of 1.8 GHz and beyond where we observe a clear speedup gain using a high bandwidth memory. Therefore, the speedup gain when deciding memory configuration, is not only decided by the application type (memory intensive or not), but also by the maximum operating frequency of the core.

Figure 1 (d) depicts which parameters of memory

Table 5. Memory sensitivity analysis

(a). Relative IPC variation when increasing memory frequency from 1333 MHz to 1867 MHz

%IPC Variation	27	21	16	9	8	8	8	7	5	4
Workloads	canneal	facesim	libquantum	H-grep	milc	omnetpp	H-sort	gemsfdtd	bwaves	H-kmeans

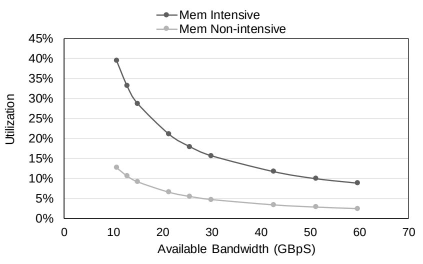
(b). Relative IPC variation when increasing memory channel from 1 to 4

%IPC Variation	35	30	28	26	21	20	14	11	10	10	4
Workloads	S-sort	canneal	facesim	milc	H-grep	libquantum	Omnetpp	gemsfdtd	astar	H-kmeans	mcf

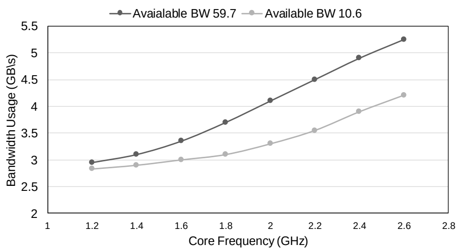
(c). Relative IPC variation when increasing memory capacity from 4 GB to 64 GB

%IPC Variation	25	22	18	10	8	7	7	7	5	5
Workloads	S-sort	S-wordcount	H-kmeans	S-nbayes	S-pagerank	H-sort	S-grep	H-grep	H-pagerank	H-wordcount

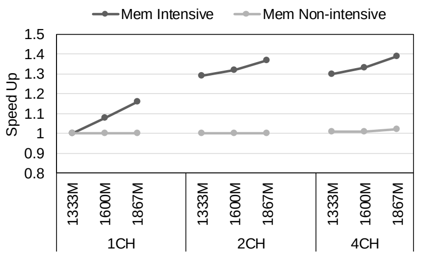
Figure 1. Bandwidth analysis



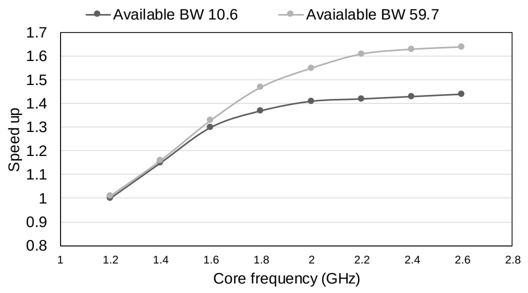
(a) Bandwidth utilization



(b) Bandwidth usage



(d) Speed up by memory



(c) Speed up by CPU frequency

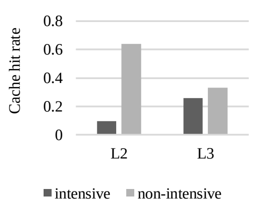
configuration help gaining more speed up when the core frequency is set to the highest; i.e. 2.6 GHz. The result shows memory configuration doesn't have any noticeable effect on the performance of memory non-intensive workloads, however memory frequency and number of channels impact the performance of memory intensive applications. In addition, the effect of memory frequency depends on the number of channels. By increasing the number of channels the influences of frequency on performance is reduced. We observe that increasing the number of channels from 2 to 4 doesn't improve the performance. This shows that the state-of-the-art server class memory controllers need to improve their management policy to effectively use 4 channels, otherwise 2 channels is sufficient for a wide range of applications studied in this work. Memory controllers utilize a large fraction of the chip transistor budget and reducing the number of channels from 4 to 2 reduces the complexity of memory controller and therefore the entire processor, without sacrificing applications performance.

3) Memory Capacity Sensitivity: Based on our results (not presented) we found that memory capacity and disk caching does not play a significant role for SPEC and PARSEC applications. However, for big data applications, due to their large input size, this is important to be investigated. To investigate the effect of memory capacity on the performance of Big Data applications, we run all workloads with 7 different memory capacities. During our experiments, Spark workloads encountered an error when running on a 4GB memory capacity due to lack of memory space for the Java heap. Hence, experiments of Spark workloads are performed with at least 8 GB of memory. Sensitivity of Spark and Hadoop applications to the memory capacity has been presented in Table 5 (c).

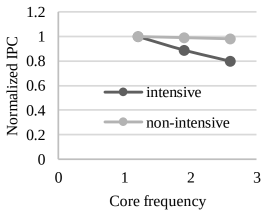
B. *Micro Architectural Analysis*

Figure 2 reports the micro architectural analysis results for the two classes of studied applications. The first parameter to study is CPU stall. It is well known that front-end stall directly incurs performance loss. Frontend stalls are also responsible for wasting power consumption. Figure 2 (d) shows stalled cycle per instruction for the two studied classes. Stalled CPI of memory sensitive workloads is almost 3 times more than non-sensitive workloads. Memory sensitive workloads suffer more from front-end stalls as the deep hierarchy of caches delays instruction–fetch and increases fetch penalty. While in general hardware prefetcher in modern multi-core processors are effective to improve performance of applications by reducing frequency of front-end stalls, for memory sensitive applications a noticeable front-end stall is still observed which indicates that a significant improvement is still needed for prefetchers.

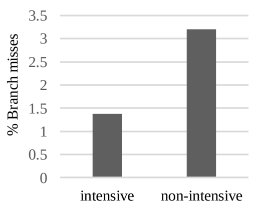
Figure 2 (a) demonstrates the L2 and L3 cache hit rates. The main difference between memory intensive and non-intensive workloads is in L2 and L3 hit rates. Memory intensive applications show a very low L2 and L3 hit rate. As an example, Canneal, which is one of the most memory sensitive applications, has L2 and L3 hit rates of 0.02 and 0.03 respectively. However, for some applications, L3 can mask L2 misses. An example is Hadoop wordcount, with a 0.41 L2 hit rate, where its L3 hit rate is 0.44, enough to prevents Hadoop wordcount performance to suffer from low L2 hit rate.



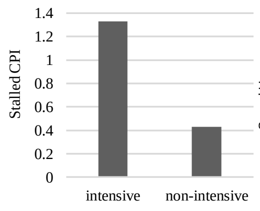
(a) Cache hit rate



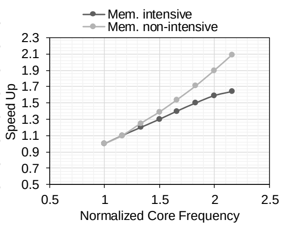
(b) IPC



(c) Branch misses



(d) Stalled CPI



(e) Speed up

Figure 2. Workloads' micro architectural behavior

Figure 2 (b) shows the average normalized IPC of memory intensive and non-intensive workloads for different core frequencies. The results show that increasing the core frequency reduces IPC of memory intensive applications. To show the effect of IPC reduction on the performance of the workloads in terms of execution time, we provide the average speed up of workloads in Figure 2 (e). The execution time results are normalized to the minimum execution time of each application. The observation shows that memory non-intensive workloads speed up gain scale linearly with core frequency as their IPC impacted by only 2%, on average. For memory-intensive workloads, the speed up curve falls below a linear curve, and even saturates when increasing the frequency beyond 2 GHz. As discussed earlier, increasing the bandwidth can mitigate this slightly. Increasing the available bandwidth from 10.6 GB to 59.7 only improves performance by 16%. Therefore, this is not an effective solution as the bottleneck exists in off chip memory access latency.

Based on Figure 3, we classify big data applications in two group of CPU-intensive and Disk-intensive class. Our decision criteria for this classification is based on the average Disk bandwidth usage. This Figure shows Spark wordcount, Spark grep, Spark PageRank, Hadoop Sort, Hadoop grep to be Disk-intensive while others to be CPU-intensive.

Workload classification. As the main goal of this paper is to study the combined impact of node architecture and cloud workload characteristics as well as performance/cost analysis, it is important to first classify those workloads. To this goal, we have explored the micro architectural behavior of studied workloads to classify those workloads and find more insights. We divided workloads into two major groups of memory intensive and memory non-intensive. Each group of memory intensive and non-intensive applications will classify to two more Hybrid groups of I/O intensive and CPU intensive. This classification will help us later to accurately formulate the relation of performance and application characteristics.

IV. Performance Cost Analysis

In this section, we formulate performance and cost analysis of different application classes in a scale-out environment. The first part of this section is devoted to formulating the total cost of ownership for different server configurations, using the cost offered by IBM/SoftLayer. We then develop equations to formulate the performance improvement of each application class with respect to the baseline configuration. These equations will be exploited by MeNa to select the most performance/cost efficient memory and CPU configuration for each class of application.

A. Cost model

In this part, we analyze the parameters that are influencing the total cost of ownership (TCO) in a data center. Our goal is to establish a relationship between performance of studied applications reported in section 3, and the total cost of ownership when running these applications. We utilize EETCO [40] to drive a model for estimating TCO. The following five main factors determine the TCO in a data center:

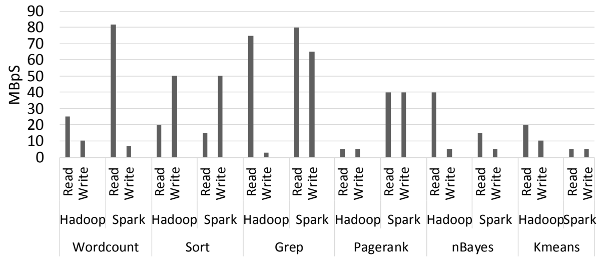


Figure 3. Disk access of Big Data applications

• *Datacenter Infrastructure Cost*: the cost of acquisition of the datacenter building (real estate and development of building) and the power distribution and cooling equipment acquisition cost. The cost of the infrastructure is amortized over 10~20 years.

• *Server Cost Expenses*: the cost of acquiring the servers, which depreciates within 3~4 years.

• *Networking Equipment Cost Expenses*: the cost of acquiring the networking equipment, which depreciates within 4~5 years.

• *Datacenter Operating Expenses*: the cost of electricity for servers, networking equipment and cooling.

• *Maintenance and Staff Expenses*: the cost for repairs and the salaries of the personnel.

$$TCO = C_{infrastructure} + C_{server} + C_{network} + C_{power} + C_{maintenance} \qquad \text{(Eq. 2)}$$

In the above equation, the first line represents the capital expenses (CAPEX) and the second line represents the operational expenses (OPEX). Given that the infrastructure, network, power, and maintenance costs are decided by the server configuration parameters, we can simplify Eq.2 with TCO = C_{server} (configuration) which indicates the total cost of ownership is a function of server configuration parameters.

The server TCO per month is determined as follows:

$$C_{server} = C_{processor} + C_{memory} + C_{Disk} \qquad \text{(Eq. 3)}$$

In this work we do not consider configuring network and disk for performance optimization. Therefore, to establish a relationship between performances of applications as well as the TCO, we are simply treating disk and network cost as constant. We extracted TCO data for 32 available server configurations in IBM\SoftLayer. We used regression technique to derive cost equation based on available server configurations. Table 6 shows the available configurations.

The per-server costs includes configurable DRAM, configurable processor and constant disk and network costs. At the server level, we account for the number of socket, core per socket, and core frequency. We then formulate each server performance as the product of per-processor performance (using data collected in our experiments) and the number of processor in each server.

The equation for monthly charge per server based on the server's processing part configuration is as follows:

$C_{processor} = Intercept + \alpha N_{socket} + \beta N_{core} + \gamma Frequency_{core}$ (Eq. 4)

Table 7 shows the results of processor cost's regression equation. For the cost of memory, we derived two different equations. The first, considers the effect of memory frequency on the cost of each memory module. The maximum capacity of each available memory module is 32 GB. This is the maximum available DRAM module in the market.

$C_{memory\ module} = [(9 \times Capacity) \times (Mem.\ Frequency - 0.31)] - 5 \times N_{channel}$ (Eq. 5)

Beyond 32 GB, the memory capacity is estimated using the following equation:

$C_{memory\ subsystem} = (1.81 \times Capacity) + 364$ (Eq. 6)

All above cost equations are the predicted charge that subscriber must pay for renting a server on a cloud. This includes the power, cooling, and maintenance related costs of the server.

B. *Performance model*

In section 3, we classified studied applications into 4 different classes. Based on our characterization and previous analysis, a set of performance equations is derived for each application class. These equations are developed using regression technique on a database collected through a comprehensive experiment presented in section 3. We formulate those observations into regression-based equations to express performance of an application as a function of processor and memory configurations. Given the influence of both processor configuration and memory configuration on performance, we divide the performance gain equation into two parts; a part showing the performance gained by processor and another one showing the performance gained by the memory subsystem. The base configuration, which was used to account for performance gain is presented in table 8.

Table 9 shows the performance gain as a function of core count. For each class, we derived two different equations as the core frequency changes the behavior of applications. Similarly, Table 10 shows the performance gain by changing the core frequency.

Performance gain as a function of memory frequency and number of channels for various classes of applications is shown in table 11. We only provide the equation for CPU-Memory intensive application class because other classes of applications do not gain noticeable performance benefit by increasing the memory frequency and the number of channels.

In addition, we derived the performance gain equation as a function of DRAM capacity as follows (only for Big Data applications, as the rest are not sensitive to DRAM capacity):

$Performance_{capacity} = 0.0018\ capacity + 0.99$ (Eq. 7)

The minimum capacity for Big Data application is also determined by the following equation:

$Minimum\ capacity = (footprint \times core) / 8$ (Eq. 8)

Table 6. Available server configuration on IBM\SoftLayer

#Socket	#Core per socket	Core frequency	Memory frequency	#Memory channel
1 - 4	2 - 16	2 - 3.6 GHz	1333, 1600, 1867 MHz	1 - 4

Table 7. Values of server cost's formula

Parameter	Intercept	α	β	ϒ	R^2
Value	-353.5	208.1	31.4	54.9	0.82

Table 8. Base server configuration

#Socket	#Core per socket	Core frequency	Memory frequency	#Memory channel	Price
1	2	2 GHz	1333MHz	1	73$

Table 9. Performance gain by increasing core count

App. class	Core frequency > 2.8	Core frequency ≤ 2.8
Mem-CPU	Perf = 0.16 core + 0.67	Perf = 0.28 core + 0.42
CPU	Perf = 0.28 core + 0.48	Perf = 0.3 core + 0.38
IO	Perf = 0.1 core + 0.79	Perf = 0.14 core + 0.7
Mem-IO	Perf = -0.02 core + 1.04	Perf = -0.01 core + 1.43

Table 10. Performance gain by increasing core frequency

App. class	Core count > 8	Core count ≤ 8
Mem-CPU	Perf = 0.04 freq + 0.96	Perf = 0.43 freq + 0.57
CPU	Perf = 0.25 freq + 0.75	Perf = 0.3 freq + 0.7
IO	Perf = 0.09 freq + 0.91	Perf = 0.26 freq + 0.74
Mem-IO	Perf = 0.03 freq + 0.93	Perf = 0.03 freq + 0.95

Table 11. Performance gain by memory frequency and channel

	High core and frequency	Low core and frequency
Memory frequency	Perf = 0.08 freq + 0.92	Perf = 0.03 freq + 0.96
Channel	Perf = 0.15 Ch + 0.89	Perf = 0.09 Ch + 0.97

In the next section, MeNa exploits these performance and cost equations to calculate performance/cost for a given user defined budget.

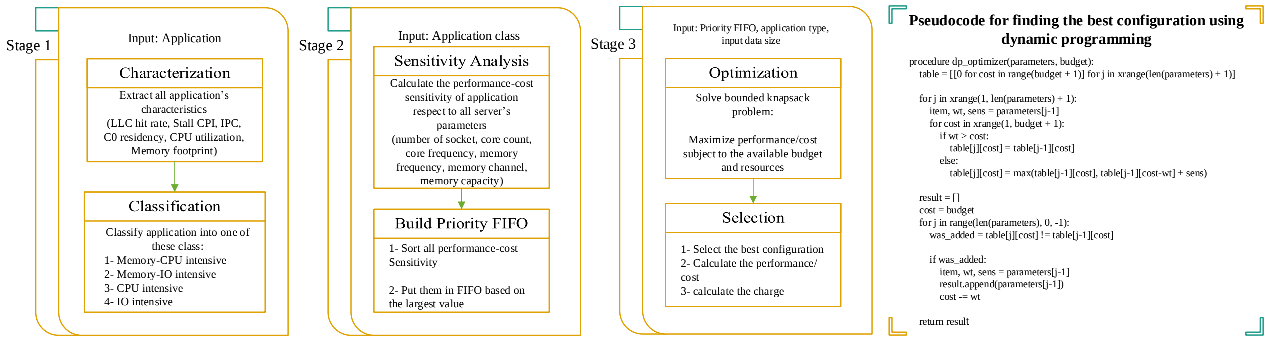
V. MEMORY NAVIGATOR (MENA)

In this section, we present our novel methodology for selecting and configuring DRAM system-level parameters in scale-out environment. Our methodology is based on the comprehensive analysis provided in previous sections.

A. *Methodology*

Figure 4 shows an overview of the proposed memory navigator. MeNa is a three-stage methodology, which navigates memory and CPU parameters to find the best performance/cost configuration for a given budget set by the user. In addition to memory parameters MeNa also navigates processor parameters as performance gain of memory subsystem is influenced by the interaction of both, as shown earlier in this paper.

The first stage of MeNa is to determine cloud applications behavior. Using the microarchitectural analysis presented earlier in section 3.2, MeNa classifies application into two main classes; memory intensive and memory non-intensive. Additionally, we divided applications to two more classes, namely CPU and I/O intensive, for more accurate performance estimate. Therefore, there are a total of four different classes as follow: 1) Mem-CPU intensive 2) Mem-IO intensive 3) CPU intensive 4) IO intensive. The classification is done on a three layer fully connected neural network trained by our training database. Figure 5 shows the first stage of MeNa. The neural network has 6 inputs and 4 outputs. Each output neuron stands for a class and it gives a probability between 0 and 1.


Stage 1
Input: Application
Characterization
Extract all application's characteristics (LLC hit rate, Stall CPI, IPC, C0 residency, CPU utilization, Memory footprint)
Classification
Classify application into one of these class:
1- Memory-CPU intensive
2- Memory-IO intensive
3- CPU intensive
4- IO intensive
Stage 2
Input: Application class
Sensitivity Analysis
Calculate the performance-cost sensitivity of application respect to all server's parameters (number of socket, core count, core frequency, memory frequency, memory channel, memory capacity)
Build Priority FIFO
1- Sort all performance-cost Sensitivity
2- Put them in FIFO based on the largest value
Stage 3
Input: Priority FIFO, application type, input data size
Optimization
Solve bounded knapsack problem:
Maximize performance/cost subject to the available budget and resources
Selection
1- Select the best configuration
2- Calculate the performance/cost
3- calculate the charge
Pseudocode for finding the best configuration using dynamic programming

```
procedure dp_optimizer(parameters, budget):
    table = [[0 for cost in range(budget + 1)] for j in xrange(len(parameters) + 1)]

    for j in xrange(1, len(parameters) + 1):
        item, wt, sens = parameters[j-1]
        for cost in xrange(1, budget + 1):
            if wt > cost:
                table[j][cost] = table[j-1][cost]
            else:
                table[j][cost] = max(table[j-1][cost], table[j-1][cost-wt] + sens)

    result = []
    cost = budget
    for j in range(len(parameters), 0, -1):
        was_added = table[j][cost] != table[j-1][cost]

        if was_added:
            item, wt, sens = parameters[j-1]
            result.append(parameters[j-1])
            cost -= wt

    return result
```

Figure 4. MeNa methodology overview

Hence, a neuron with the highest value determines the class of application.

To identify the cost to increase performance by changing each server parameters, we define a quantity called performance-cost sensitivity. For example, the performance-cost sensitivity to the number of cores per processor is defined as follow:

$$\text{Sens (core)} = ((\partial\ \text{Performance})/(\partial\ \text{core}))/((\partial\ \text{Cost})/(\partial\ \text{core})) \quad \text{(Eq. 9)}$$

The second stage is to calculate this quantity with respect to the number of sockets per server, number of cores per processor, core frequency, memory frequency, number of memory channels, and the capacity of DRAM. The equations for performance are provided in tables 9, 10, 11, as well as equations 7, and 8. Cost equations are reported in equations 4, 5, and 6. This quantity helps MeNa to set a priority for each parameter when allocating processor and memory resources. In this step, MeNa sorts all sensitivity results and based on the largest results it puts them into a FIFO. We refer to this as Priority FIFO.

In the third stage, MeNa determines the configuration to maximize the performance/cost while satisfying the subscriber budget. For this purpose, MeNa solves the following problem known as bounded knapsack by using dynamic programming:

$$\text{Maximize } \sum_{i=1}^{n} Perf_i\ Conf_i$$

$$\text{Subject to } \sum_{i=1}^{n} Cost_i\ Conf_i \ \leq Budget \text{ and } \min_i \leq \text{Conf}_i \leq \max_i$$

Where Confi represents the number or the value of parameter i, mini and maxi are the minimum and the maximum available resource for parameter i. Also, Costi present the cost corresponding to Confi (Calculated by the cost model in section 4.1). Similarly, Perfi present the performance improvement corresponding to Confi (Calculated by the performance model provided in section 4.2).

The last step determines the final configuration and its corresponding performance/cost ratio as well as the corresponding cost using performance-cost equation.

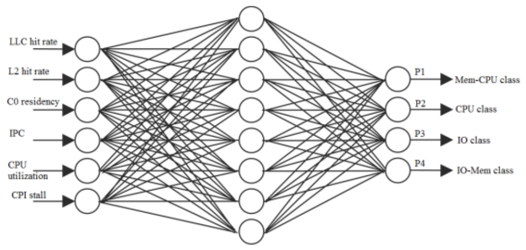


Figure 5. MeNa classifier (Neural Network)

B. Validation

To show how MeNa allows subscribers to intelligently search all server configuration space for finding the best parameters to maximize performance/cost while meeting the user specified budget, we validate MeNa against an oracle configuration identified through the brute force search. We apply the brute force search as follow: First, we select an application and set a budget. Then we find all configurations that achieve cost equal or smaller than the target budget. Finally, we run the application on those configurations and calculate its performance gain compared to the base configuration. By knowing the cost of each configuration, we calculate performance/cost for each configuration. The best configuration is referred as the oracle configuration. We then use MeNa to find the best configuration for the same application. Comparison between MeNa's outcome and oracle outcome shows that MeNa methodology can find the best configuration with 9% performance/cost error rate on, average for our training data sets. In the worst case, a 17% error is a small price for avoiding an exhaustive brute-force search. The standard deviation of errors is 4%. Table 12 shows the error rate of MeNa compared to oracle configuration for three test applications from our training data sets.

Figure 6 shows the average performance/cost error rate of MeNa for various budgets. This result shows MeNa accuracy is higher for Mem-CPU class (8% error rate) while it has the lowest accuracy for I/O intensive class (12%). Moreover, MeNa is more accurate for mid-range target budget (between

Table 12. MeNa Validation

Application	Class	Budget	Configuration	#core	Core freq.	#Socket	Mem. Cap.	Mem. Freq.	#channel	Perf./Cost	Error
Spark Kmeans	Mem-CPU intensive	500$	Oracle	12	2.6	1	16	1333	2	0.977268	3.5%
			MeNa	13	2.8	1	12	1333	4	0.942737	
Hadoop Sort	IO intensive	180$	Oracle	4	2.8	1	4	1333	1	0.885733	10%
			MeNa	5	3	1	2	1333	1	0.796636	
Hadoop WordCount	CPU intensive	400$	Oracle	8	3	1	6	1333	1	1.010769	8.6%
			MeNa	10	2.8	1	8	1333	2	0.923691	

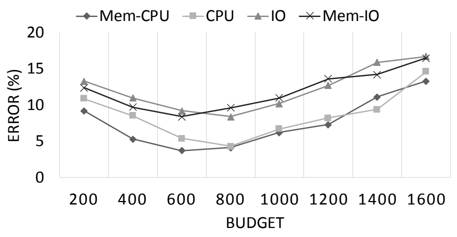


Figure 6. Average error for different class of applications and Budget

500$ and 900$).

C. Evaluation

In this section, we evaluate MeNa with unknown applications for various target budget. The selected applications are: Hadoop Terasort, Hadoop Scan, Spark sort, and Spark Nweight. Table 13 shows the features of each application and the class identified by MeNa. Tables 14, 15, 16, and 17 show the configurations selected by MeNa for the given budgets. We also report the performance/cost error rate of each application for each target budget. The results show that MeNa identifies a configuration with performance/cost of on average 11% close to an oracle configuration.

The evaluation of MeNa can also be used to derive architectural insights for server designers. For instance based on MeNa results we can see CPU intensive applications to demand large number of cores and low frequency processors while I/O intensive applications to require low number of core but high frequency processors. The result of I/O-Memory intensive application shows a very high performance/cost ratio since all options that can improve the performance are playing against each other. As a memory intensive application, we need large number of cores and high core frequency to take advantage of a fast memory subsystem. As an I/O intensive application, however, increasing the number of cores and core frequency exacerbate the I/O accesses and impacts application's performance. These findings have been corroborated by our characterization's result presented earlier in section 3. Another observation is that increasing the memory frequency does not enhance performance/cost ratio even for memory intensive applications. On the other hand, increasing the number of memory channels improve the performance/cost ratio of memory intensive applications.

The trend in figure 7 shows that, regardless of application class, scale-out approach cannot always enhance the performance/cost unless the scale-up solution is exploited. For example, increasing the number of sockets, when the number of cores is low, reduces performance/cost. However, when the number of cores increases, performance/cost enhances by increasing the number of sockets. This means that the unseen cost that subscribers pay for a server such as for cooling, maintenance, and network, forces them to get the maximum utilization from their server. To show how memory frequency and number of channels affect the performance/cost of applications, we present the average results of our dataset for different number of cores in figure 8. Figure 8 shows that increasing the memory frequency has almost no impact on improving the performance/cost ratio. On the other hand, increasing the number of channel improves performance/cost ratio. However the improvement will be diminished if the number of cores is low. In fact, as long as the number of cores is high, enough pressure is being put on the memory subsystem, therefore results in enhancing the performance when the number of memory channels increases.

Table 13. Applications' features

Application	LLC hit	L2 hit	C0 residency	CPU util.	CPI stall	IPC	Class
S-Nweight	32	39	89.27	83	2.7	1.6	CPU-Mem
H-Terasoer	44	41	74.88	75	0.26	2	CPU
H-Scan	69	59	29.23	37	0.42	0.72	IO
S-srt	40	35	51.28	48	0.43	0.8	IO-Mem

Table 14. Configurations selected for Spark Nweight

Budget	250$	450$	700$	900$	1100$	1400$
#Core	7	13	13	6	12	16
Core_freq (GHz)	2.8	2.8	2.8	2.8	2.8	2.8
#Socket	1	1	2	4	4	4
Mem_cap (GB)	5	10	16	16	24	32
Mem-freq (MHz)	1333	1333	1333	1333	1333	1333
#channel	1	2	2	1	2	4
Perf/Cost	0.826133	0.765555	1.01344	0.792148	1.160709	1.53858
Error (%)	8.2	5.3	3.1	5.6	7.9	11.1

Table 15. Configurations selected for Hadoop Terasort

Budget	250$	450$	700$	900$	1100$	1400$
#Core	7	6	6	6	12	16
Core_freq (GHz)	2.8	2.8	2.8	2.8	2.8	2.8
#Socket	1	2	3	4	4	4
Mem_cap (GB)	3.5	6	9	12	24	32
Mem-freq (MHz)	1333	1333	1333	1333	1333	1333
#channel	1	1	1	1	1	1
Perf/Cost	1.119605	1.093411	1.077358	1.069507	1.589477	1.840536
Error (%)	9.8	7.5	5.9	6	9.4	10.8

Table 16. Configurations selected for Hadoop Scan

Budget	250$	450$	700$	900$	1100$	1400$
#Core	5	5	5	4	9	9
Core_freq (GHz)	3.6	3.6	3.6	3.6	3.6	3.6
#Socket	1	2	3	4	4	4
Mem_cap (GB)	2.5	5	7.5	8	18	18
Mem-freq (MHz)	1333	1333	1333	1333	1333	1333
#channel	1	1	1	1	1	1
Perf/Cost	1.046293	1.044781	1.044279	0.922948	1.449773	1.449773
Error (%)	14.6	14.8	12.2	13.5	15.4	17.2

Table 17. Configurations selected for Spark sort

Budget	250$	450$	700$	900$	1100$	1400$
#Core	3	3	2	2	4	4
Core_freq (GHz)	3.6	3.6	3.6	3.6	3.6	3.6
#Socket	1	2	3	4	4	4
Mem_cap (GB)	3	6	6	8	16	16
Mem-freq (MHz)	1333	1333	1333	1333	1333	1333
#channel	1	2	2	2	4	4
Perf/Cost	0.510734	0.455947	0.464989	0.445342	0.456145	0.364948
Error (%)	13.3	14.7	9.4	10.1	16	15.3

Another interesting observation is that allocating higher budget for an application does not necessary yield a better performance/cost, shown in figure 9. This implies that there needs to be a method to enable subscribers to provision a rational budget for their application to get the max performance/cost benefits. MeNa proved that it is an answer for such urgent demand. MeNa methodology is architecture independent and therefore it can still be utilized for future technology.

VI. Related Work

This section summarizes the relevant literature on characterization and optimization of memory subsystem and cloud platforms.

A. Cloud

Jackson et al. [12], Barker et al. [13], Vecchiola et al. [14], and Farley [15] analyzed the HPC applications, latency-

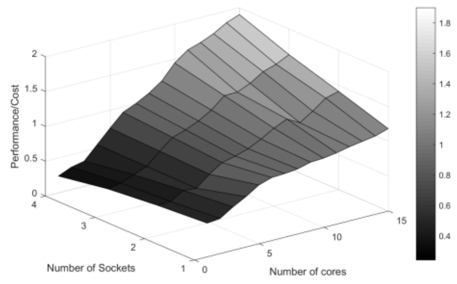


Figure 7. Average Performance/Cost correspond to the average number of core and socket

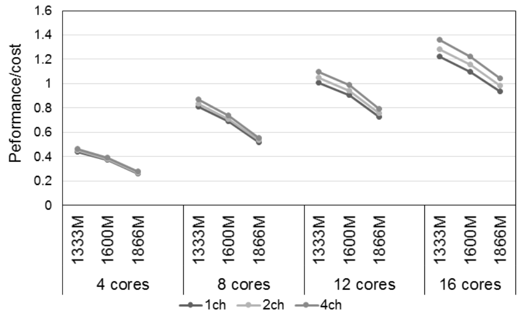


Figure 8. Effect of memory parameters on Performance/Cost

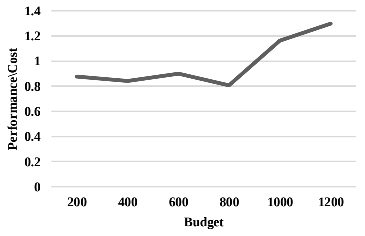


Figure 9. Average Performance/Cost correspond to Budget

sensitive applications, scientific applications, and micro-benchmark applications, respectively, on the cloud. Ferdman et al. [16], Kanev [39], and Kozyrakis et al. [17] analyzed cloud-scale workloads to provide infrastructure-level insights. Yang et al. [18] compared public cloud providers from performance and cost perspectives. Blem et al. [19] compared two different ISAs for various cloud scale applications. Guevara et al. [20] studied how heterogeneous platforms bring energy-efficiency for cloud applications. None of these studies have focused on the influence of memory subsystem and its parameters on the performance, power and cost in cloud.

B. *Memory*

Dimitrov et al. [21] characterized the memory access patterns of Hadoop and noSQL big data workloads. However, their work hasn't examined the impact of memory parameters on the performance and power consumption of the system. Clapp et al. [22] provides a performance model that considers the effect of memory bandwidth and latency for big data, high performance, and enterprise workloads. Alzuru et al. [23] investigates how Hadoop workload demands hardware resources such as high-end memory. Our observation appears to contradict the conclusion of their work claiming that optimal memory capacity for Hadoop workloads is 96 GB. Zhu et al. [24] evaluates contemporary multi-channel DDR SDRAM and Rambus DRAM systems in SMT architectures and proposed a thread-aware DRAM optimization technique. Basu et al. [25] focuses on page table and virtual memory optimizations for big data workloads and Jia et al. [26] characterized cache hierarchy for a Hadoop cluster. Moreover, several studies have focused on memory characterization of SPEC benchmark suites [27, 28, 29, 30]. Hajkazemi et al. explored the performance of Wide I/O and LPDDR memories [31] and proposed an adaptive bandwidth management for HMC+DDR memory [32] without considering the cost of memory configuration and big data applications.

C. *Big Data*

Pan et al. [33] selected four big data workloads from the BigDataBench to study their I/O characteristics. Liang et al. [34] characterize the performance of Hadoop and DataMPI workloads using Amdahl's second law. Beamer et al. [35] analyzes the performance characteristics of three high performance graph analytics. They found that graph workloads fail to fully utilize the platform's memory bandwidth. A recent work [36] used Principle Component Analysis to identify important characteristics of BigDataBench workloads. Results of Jiang work [37] show that Spark workloads have different behavior than Hadoop and HPC benchmarks. Issa et al. [38] studied performance characterization of Hadoop K-means iterations. Based on the characterization results they propose a model to estimate the performance of Hadoop K-means iterations. Malik et al. characterized Hadoop on big-little cores and microservers without extracting the performance model [41, 42, 43, 44]. Neshatpour et al. analyzed the performance of big data applications on heterogeneous architecture [45] and accelerated Hadoop applications' performance using FPGA [46, 47].

VII. CONCLUSION

Main memory performance is becoming an increasingly important factor contributing to overall system performance and the operational and capital costs. This particularly becomes important for server-class architectures as more applications are moving to the clouds. This suggests that it is important to understand the role of memory configuration parameters, such as capacity, number of channels, and operating frequency, for performance and energy-optimization of emerging class of applications in scale-out environment. In response, this work addresses these challenges with a real-system experimental setup. Our analysis reveals several interesting trends and provides key system and architectural insights on how memory configuration parameters must be tuned across various classes of applications to achieve high performance at low cost. To the best of our knowledge, this is the first work that provides a methodology for improving the performance/cost ratio of server class architectures in a scale-out environment while considering the memory parameter as well as processor parameters. We proposed a novel three-stage methodology to navigate memory parameter referred as MeNa. MeNa uses a fully connected Neural Network to characterize and classify applications. Based on the characterization results, we present experimentally derived models for estimating and predicting the impact of memory and processor parameters on capital and operational cost and performance of applications. MeNa uses those models to navigate memory and processor configuration parameters in order to find the best configuration to maximize performance/cost ratio for a given user defined budget. MeNa utilizes dynamic programming to solve bounded knapsack problem to achieve that goal. The validation results on our extensive database show MeNa to have 91% accuracy on average to estimate the performance/cost ratio compared to a brute force approach. MeNa enables subscribers to provision a rational budget for their application to get the max performance/cost in cloud environment. MeNa also reveals several interesting trends and provides key insights that can be leveraged by server designers for various optimization goals.

REFERENCES

[1] A. Michael et al., “A view of cloud computing,” *Communications of the ACM* 53, no. 4, pages 50-58, 2010.

[2] Luis Columbus, “Roundup of Cloud Computing Forecasts and Market Estimates,” 2015. URL: http://www.forbes.com/sites/louiscolumbus/2015/01/24/roundup-of-cloud-computing-forecasts-and-market-estimates-2015/#6fbeb58e740c

[3] J. Gelas, “Server Buying Decisions: Memory,” URL: http://www.anandtech.com/print/7479/server-buying-decisions-memory.

[4] S. Kshitij et al., “Optimizing datacenter power with memory system levers for guaranteed quality-of-service,” *In 21st PACT*, pages 117-126, 2012.

[5] Gottscho et al, “X-Mem: A Cross-Platform and Extensible Memory Characterization Tool for the Cloud,” variations 40, no. 41: 42.

[6] W. Bircher et al., “Complete system power estimation using processor performance events,” *IEEE Transactions on Computers* 61, no. 4 pp. 563-577, 2012.

[7] X. Fan et al., “Power provisioning for a warehouse-sized computer,” *In ACM SIGARCH Computer Architecture News*, vol. 35, no. 2, pages 13-23. ACM, 2007.

[8] L. Wang et al., “Bigdatabench: A big data benchmark suite from internet services,” *In IEEE 20th HPCA*, pages 488-499, 2014.

[9] J. Henning, “SPEC CPU2006 benchmark descriptions,” *ACM SIGARCH Computer Architecture News* 34, no. 4, pages 1-17. 2006.

[10] Ch. Bienia et al., “The PARSEC benchmark suite: characterization and architectural implications,” In *PACT*, pp. 72-81, 2008.

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

[12] K. Jackson et al., “Performance analysis of high performance computing applications on the amazon web services cloud,” *In CloudCom*, pp. 159-168, 2010.

[13] S. Barker et al., “Empirical evaluation of latency-sensitive application performance in the cloud,” *In SIGMM conference on Multimedia systems*, pp. 35-46, 2010.

[14] Ch. Vecchiola et al., “High-performance cloud computing: A view of scientific applications,” *In IEEE 10th I-SPAN*, pages 4-16, 2009.

[15] B. Farley et al., “More for your money: exploiting performance heterogeneity in public clouds,” *In ACM Cloud Computing*, p. 20. ACM, 2012.

[16] M. Ferdman et al., “Clearing the clouds: a study of emerging scale-out workloads on modern hardware,” *In ACM SIGPLAN Notices*, vol. 47, no. 4, pp. 37-48, 2012.

[17] Ch. Kozyrakis et al., “Server engineering insights for large-scale online services,” *IEEE micro 30*, no. 4, pages 8-19, 2010.

[18] A. Li et al., “CloudCmp: comparing public cloud providers,” *In 10th ACM SIGCOMM conference on Internet measurement*, pp. 1-14, 2010.

[19] E. Blem et al., “A detailed analysis of contemporary arm and x86 architectures,” *UW-Madison Technical Report*, 2013.

[20] M. Guevara et al., “Navigating heterogeneous processors with market mechanisms,” *In IEEE HPCA*, pp. 95-106, 2013.

[21] M. Dimitrov et al., “Memory system characterization of big data workloads,” *In IEEE Big Data*, pp. 15-22, 2013.

[22] R. Clapp et al., “Quantifying the Performance Impact of Memory Latency and Bandwidth for Big Data Workloads,” *In IISWC*, pp. 213-224, 2015.

[23] I. Alzuru et al., “Hadoop Characterization,” *In Trustcom/BigDataSE/ISPA*,vol. 2, pages 96-103, 2015.

[24] Z. Zhu et al., “A performance comparison of DRAM memory system optimizations for SMT processors” *In 11th HPCA*, pages 213-224, 2005.

[25] A. Basu et al., “Efficient virtual memory for big memory servers,” *In ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 237-248, 2013.

[26] Zh. Jia et al., “Characterizing data analysis workloads in data centers,” *In IISWC*, pp. 66-76, 2013.

[27] L. Barroso et al., “Memory system characterization of commercial workloads” *ACM SIGARCH Computer Architecture News* 26, no. 3, 1998.

[28] A. Jaleel, “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.

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

[30] Y. Shao et al., “ISA-independent workload characterization and its implications for specialized architectures” *In ISPASS*, pp. 245-255, 2013.

[31] M. Hajkazemi et al., “Wide I/O or LPDDR? Exploration and analysis of performance, power and temperature trade-offs of emerging DRAM technologies in embedded MPSoCs,” *In ICCD*, pp. 62-69, 2015.

[32] M. Hajkazemi et al., “Adaptive bandwidth management for performance-temperature trade-offs in heterogeneous HMC+ DDRx memory,” *In GLSVLSI*, pp. 391-396, 2015.

[33] F. Pan et al., “I/O characterization of big data workloads in data centers,” In Workshop on Big Data Benchmarks, Performance Optimization, and Emerging Hardware (Springer International Publishing), pp. 85-97, 2014.

[34] F. Liang et al., “Performance characterization of hadoop and data mpi based on amdahl's second law,” *In NAS*, pp. 207-215, 2014.

[35] S. Beamer et al., “Locality exists in graph processing: Workload characterization on an Ivy Bridge server,” *In IISWC*, pp. 56-65, 2015.

[36] Zh. Jia et al., “Characterizing and subsetting big data workloads,” *In IISWC*, pp. 191-201, 2014.

[37] T. Jiang et al., “Understanding the behavior of in-memory computing workloads,” *In IISWC*, pp. 22-30, 2014.

[38] J. Issa, “Performance characterization and analysis for Hadoop K-means iteration,” *Journal of Cloud Computing* 5, no. 1, 2016.

[39] S. Kanev et al., “Profiling a warehouse-scale computer,” *In ISCA*, pp. 158-169, 2015.

[40] D. Hardy et al., “EETCO: A tool to estimate and explore the implications of datacenter design choices on the tco and the environmental impact,” *In Micro*-44, 2011.

[41] M. Malik et al., “Big vs little core for energy-efficient Hadoop computing,” *In DATE*, pp. 1480-1485, 2017.

[42] M. Malik et al., “Characterizing Hadoop applications on microservers for performance and energy efficiency optimizations,” *In ISPASS, pp.* 153-154, 2016.

[43] M. Malik et al., “System and architecture level characterization of big data applications on big and little core server architectures,” *In IEEE Big Data*, pp. 85-94, 2015.

[44] M. Malik et al., “Big data on low power cores: Are low power embedded processors a good fit for the big data workloads?,” *In ICCD*, pp. 379-382, 2015.

[45] K. Neshatpour et al., “Big data analytics on heterogeneous accelerator architectures,” *In CODES+ ISSS*, pp. 1-3, 2016.

[46] K. Neshatpour et al., “Energy-efficient acceleration of big data analytics applications using fpgas,” *In IEEE Big Data*, pp. 115-123, 2015.

[47] K. Neshatpour et al., “Accelerating big data analytics using fpgas,” *In FCCM*, pp. 164-164, 2015.