Co-Locating and Concurrent Fine-Tuning MapReduce Applications on Microservers for Energy Efficiency

Maria Malik[1], Dean M. Tullsen[2], Houman Homayoun[2]
[1]Department of Electrical and Computer Engineering, George Mason University, {mmalik9, hhomayou}@gmu.edu
[2]Department of Computer Science and Engineering, University of California San Diego, tullsen@cs.ucsd.edu

Abstract

Datacenters provide flexibility and high performance for users and cost efficiency for operators. However, the high computational demands of big data and analytics technologies such as MapReduce, a dominant programming model and framework for big data analytics, mean that even small changes in the efficiency of execution in the data center can have a large effect on user cost and operational cost. Fine-tuning configuration parameters of MapReduce applications at the application, architecture, and system levels plays a crucial role in improving the energy-efficiency of the server and reducing the operational cost. In this work, through methodical investigation of performance and power measurements, we demonstrate how the interplay among various MapReduce configurations as well as application and architecture level parameters create new opportunities to co-locate MapReduce applications at the node level. We also show how concurrently fine-tuning optimization parameters for multiple scheduled MapReduce applications improves energy-efficiency compared to fine-tuning parameters for each application separately. In this paper, we present Co-Located Application Optimization (COLAO) that co-schedules multiple MapReduce applications at the node level to enhance energy efficiency. Our results show that through co-locating MapReduce applications and fine-tuning configuration parameters concurrently, COLAO reduces the number of nodes by half to execute MapReduce applications while improving the EDP by 2.2X on average, compared to fine-tuning applications individually and run them serially for a broad range of studied workloads.

Keywords Co-locate, Tuning, MapReduce, Energy-Efficiency, Power

1. Introduction

Datacenters are the computer platforms of choice to process diverse applications in the emerging domain of big data. With the significant increase in the volume of data to process big data applications, hyperscale datacenters have gained interest as a promising computing architecture that is designed to provide a massively scalable computer architecture. Recent improvements in the networking, storage, energy-efficiency and infrastructure management [46, 47] has made hyperscaling a preferable approach to respond to the challenges associated with big data. However, introducing more nodes to existing infrastructural creates challenges for datacenters providers to balance computational power and energy efficiency. In addition, the cost of hyperscale data centers is one of the major limiting factors. To address these challenges, many recent works address the need for hardware specialized accelerators [49] to increase the performance using a fewer number of nodes [47]. However, specialized accelerator reduces the preferable homogeneous computing environment in datacenters and increases the compatibility issues for the target big data workloads that are diverse in nature and are changing at a rapid rate. In addition, the cost of deploying an accelerator in server and the operational cost of the datacenters can become highly exposed to the cost of these very high-demand applications, whether that cost is absorbed by the owner of the datacenter or passed on to a user running applications. As energy consumption and cooling cost are a major part of operational cost, hardware design priority is shifting from a performance-centric to an energy efficiency centric design methodology for server class architectures. Microservers represent an attractive microarchitecture in data centers, employing embedded-class low power processors as the main processing unit. These platforms can enhance energy efficiency and reduce operational cost. Therefore, microserver-based architectures have been proposed as an alternative to traditional high performance architectures to process big data applications [1, 28, 29, 32, 33, 34].

In this paper, we evaluate co-locating of MapReduce applications on microserver to reduce the number of nodes required in a cluster to process MapReduce applications for maximum energy efficiency. This is achieved while maintaining the cost-efficient homogeneous computing environment in the datacenters. Many big data applications rely on the MapReduce programming model and framework to perform their analysis on large-scale datasets [2, 3, 17]. MapReduce configuration parameters, as well as application and architectural parameters, directly affect its performance and energy efficiency that creates the opportunities for co-locating MapReduce applications at the node level. A closest work to ours is Bubble-up [30] and Bubble-Flux [41], where they introduce a characterization and profiling methodology and predicts the performance degradation between pairwise application co-locations. Co-locating traditional desktop and parallel applications and tuning the underlying processor (such as adapting the voltage and frequency [35, 38]) has been well studied in the literature [8, 12, 13, 14]. However, MapReduce applications, such as Hadoop-based, has fundamentally different microarchitectural behavior than traditional applications highlighted in recent work [28, 29, 48], while having significantly more tuning optimization knobs.

For MapReduce applications, it is important to evaluate which resources (CPU utilization, memory footprint, I/O read and write, etc.) are bottlenecks and how system-level (number of mappers running simultaneously in a compute node, HDFS block size), application-level (application type and input data size) and architectural-level (operating voltage and frequency) tuning parameters affect the performance, power, and energy-

efficiency. While several recent works [42, 43] show how tuning individual or a subgroup of tuning parameters at a time improves performance or energy-efficiency, they have ignored the interplay among all of these parameters at various level of abstractions. In addition, while all of the prior work mainly focused on fine-tuning optimization parameters for individual applications and in isolation, they have not studied opportunities for co-optimizing these tuning parameters for multiple scheduled applications, simultaneously.

In the presence of these optimization opportunities, a key research question is to determine the best tuning parameters at the system, application, and architecture levels that create the possibility to co-locate MapReduce applications at the node level and still maintain the energy efficiency. To this goal, we examine the impact of application, system, and architectural tuning parameters and the interplay among them on the performance and energy efficiency for various MapReduce applications. In addition, we compare two optimization strategies; Individually-Located Application optimization (ILAO) which represents a conventional approach, and Co-Located Application Optimization (COLAO) which represents running and tuning applications concurrently at a node level. ILAO tunes optimization parameters for each application individually. COLAO tunes optimization parameters concurrently to determine the best tuning parameters for maximum energy-efficiency.

To the best of our knowledge, this is the first experimental work that addresses the challenges of concurrent fine-tuning and co-locating MapReduce applications for energy efficiency. In this paper, our analysis helps to determine how critical it is to jointly fine tune system, application and architecture level parameters for maximum energy-efficiency for multiple scheduled MapReduce applications concurrently, and how fine tuning these parameters creates new opportunities for co-locating them at the node level.

The rest of this paper is organized as follows. Section 2 presents the experimental setup details. Section 3 presents the characterization analysis of MapReduce applications. Section 4 discusses the performance and energy-efficiency analysis of MapReduce applications by fine-tuning the systems, architectural as well as application level parameters. Co-located applications at node level analysis is discussed in section 5. Section 6 presents the evaluation on scalability of COLAO technique. Section 7 provides the related work. Finally, section 8 presents the concluding remarks.

2. Experimental Setup

This section describes our hardware and software platforms used to run real experiments on reasonable server hardware, studied applications and the tuning parameters, our measurement methodology, and tools used to enhance our results analysis.

2.1 Hardware/software infrastructure

We conduct our study on an 8-node cluster comprised of Intel Atom C2758 CPUs. Each Intel Atom has 8 processor cores per node and a two-level cache hierarchy with 8GB of system memory using DDR3 @1600MHz**.**The operating system is Ubuntu 13.10 with Linux kernel 3.11 and Hadoop version 2.6.1. For this study, we have focused on the parameters that are system configurable and are transparent at the user level, namely HDFS block size, input data size per node, number of mappers, and the operating frequency of the processor. While there are more tuning parameters to be included, this paper attempts to provide an in-depth understanding of how concurrent tuning of the studied parameters at various levels can impact the performance and energy efficiency. The buffer page caches are flushed at the start of each run to ensure that data is read fresh from HDFS.

2.2 Application Diversity

A Hadoop MapReduce cluster can host a variety of big data applications running concurrently. We have included 11 widely used Hadoop applications in this research. Out of these, four applications are Hadoop micro-benchmarks that are used as kernels in many Big Data applications, namely Wordcount-WC, Sort-ST, Grep-GP and TeraSort-TS in this paper. We have also included seven real-world applications namely Naïve Bayes (NB), FP-Growth (FP), Collaborative Recommendation Filtering (CF), support vector machine (SVM), PageRank (PR), Hidden Markov Model (HMM), and KMeans (KM) [18].

2.3 Input Data Size

The size of data can have significant impact on microarchitectural behavior [31]. For this research, we therefore use three input data sizes per node for each application; 1GB, 5GB, and 10GB representing small, medium and large data sets. For instance, 10GB input data size per node presents 80GB input data size processed by application in an 8-node cluster.

2.4 Interdependent Tuning Parameters

We have studied the impact of the system, application, and architectural level tuning parameters including the HDFS block size (64MB, 128MB, 256MB, 512MB, 1024MB), the number of mappers that run simultaneously on a single node (1-8), and frequency settings (1.2GHz, 1.6GHz, 2.0GHz, 2.4GHz) to evaluate how these parameters affect energy efficiency.

2.5 Measurement

We use Perf [4] to capture the performance characteristics of the studied applications. Perf is a Linux profiler tool that records the hardware performance counters. Perf exploits the Performance Monitoring Unit (PMU) to measure performance as well as other hardware events accurately. Perf multiplexes the PMUs, therefore, to obtain accurate values for several hardware events, we run each workload multiple times.

For measuring power consumption, Wattsup PRO power meter is used [5]. It measures and records power consumption at one second granularity. The power reading is for the entire system, including core, cache, main memory, hard disks and on-chip communication buses. We have collected the average power consumption of the studied applications and subtracted the system idle power to estimate the dynamic power dissipation of the entire system. The same methodology is used in [43], for power and energy analyses. Dstat [26] is used for main memory, disk and CPU utilization analysis. Dstat is a system-monitoring tool, which collects various statistics of the system**.**

3. MapReduce Applications Characterization

In this section we characterize MapReduce applications by monitoring the real time system resources as well as micro-architectural metrics to understand their runtime behavior and resource utilization. This analysis helps us to generalize the optimal configuration parameters with respect to the application type.

3.1 Resource Utilization Analysis

To explore the resource utilization of MapReduce applications, we collect the following metrics:

- **CPU utilization.** The dstat profiling tool classifies CPU utilization into different types such as CPUuser, CPUidle, CPUiowait, etc. We collect the data for CPUuser utilization *which represent CPU usage by a user (usr) processes* - and CPUiowait which represents the percentage of time CPU is idle waiting for I/O operation to complete.
- **I/O read/write Bandwidth,** which reports the disk I/O bandwidth rate.
- **Memory Footprint**, which reports the minimum amount of memory (in KB) required to run the application. Additionally, the MemCache metric shows the amount of file contents kept in the cache that are yet to be written to the disk.

In addition, we have included several micro-architectural parameters including, IPC, Instruction Cache Misses per Kilo instructions (MPKI), LLC MPKI, and Branch Misprediction rate.

3.2 PCA and Clustering Analysis

Unfortunately, there is no single perfect hardware counter that accurately indicates performance behavior of an application. There is substantial debate about what hardware counter event can accurately indicate performance across a variety of applications [17, 29, 31]. In this paper several micro-architectural metrics and runtime resource utilization metrics are collected and are used in identifying MapReduce application characteristics. However, collecting all of the performance counter data requires multiple runs because the counter resources are multiplexed in the microserver. In order to avoid multiple runs, we would like to identify a minimal set of counters that can be collected in a single run, maximizing correlation with performance, while minimizing redundant counters (correlated to each other). These should be representative of application, software stack, and micro-architecture interactions in the presence of various system calls.

A systematic approach for this purpose is to use Principal Component Analysis (PCA). PCA analysis allows us to monitor the most vital and distinct micro-architecture parameters to capture application characteristics. PCA captures most of the data variation by rotating the original data to a new variable in a new dimension, commonly known as the principal components (PC). These new variables are uncorrelated to each other and are a linear combination of the original data. We employ PCA to project our 14 original gathered features into a new dimensional space to determine the most important features along different PC dimensions. The number of PCs can be less than or equal to the number of original data variables. We only present the first two PCs covering 85.22% of the total variance due to space limitation. PCA is sensitive to the relative scaling of the original variables. Thus, we have normalized the data to the unit normal distribution for segregating the impact of the variable range of each feature metric. Figure 1 shows the scatter plot of the first and second principal components, PC1 and PC2. Features that appear closer in this Figure typically exhibit similar behavior.

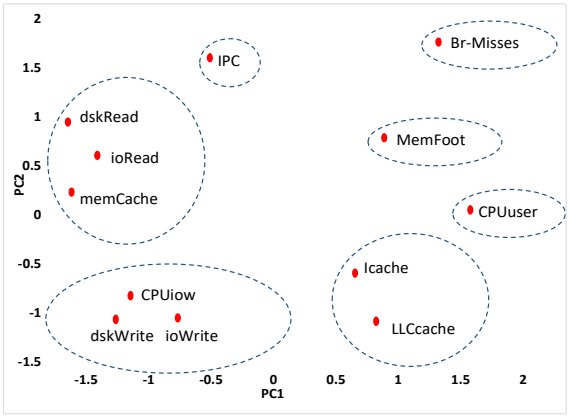

Figure 1: Scatter plot of feature metrics using first and second principal components

Later, we apply a hierarchical clustering technique to group similar features and finally analyze the results as shown in the Figure 1. We have reduced the features to 7 most important and distinct ones that are CPUuser, CPUiowait, I/O Read, I/O write, IPC, Memory Footprint, LLC MPKI to characterize the MapReduce applications. Based on these resource profiling and micro-architectural characteristics, the applications are characterized into compute-bound (C), combination of compute-bound and I/O-bound referred to as hybrid (H), memory-bound (M) and I/O-bound (I) classes. We observe (details are presented in Section 4) that the optimal configuration parameters for maximum efficiency are highly correlated to application type (I/O bound, compute-bound, memory-bound or hybrid), which can be identified by underlying microarchitectural behavior.

4. Fine-Tuning MapReduce Applications for Eenergy-Efficiency

In this section, we discuss and analyze the experimental results of the MapReduce applications at a single node level on an Atom microserver, across a wide range of Hadoop configuration parameters. Hadoop MapReduce performance and energy efficiency is sensitive to many configuration and system parameters; however, we focus on the parameters that are system configurable and transparent to the user space, configurable at the user level. This analysis helps to determine how critical it is to jointly fine tune system, application and architecture level parameters for maximum energy-efficiency for multiple scheduled MapReduce applications concurrently, and how fine tuning these parameters creates new opportunities for co-locating them at the node level.

4.1 Execution Time Analysis

Figure 2 (represented as a bar graph) shows the execution

time of MapReduce applications with respect to the number of mapper slots, in brief mappers, HDFS block size and operating frequency with a fixed input data size of 10GB per node. We have performed the experiments for the input data size of small, medium, and large with all mappers (ranges from 1 to 8), however due to space limitation and graph readability, Figure 2 presents the results for selected applications with 1, 4 and 8 mappers at the large input data size. Across almost all studied applications, the HDFS block size of 64MB -- the default HDFS block size -- has the highest execution time. Small HDFS block size generates a large number of map tasks [number of map task = Input data size /HDFS block size], which increases the interaction between master and slave nodes. These interactions are necessary to request the HDFS block location information. On the other hand, large HDFS block size reduces the slave node interaction with the master node. Additionally, with a large block size, less metadata is required to be stored on the master node, which can more likely be placed in the memory, which is faster to access. Conversely, storing large chunks of data on a node can create a performance bottleneck if the application accesses the same data repeatedly. This behavior explains the parabolic behavior in the compute-bound and hybrid applications such as *Wordcount*, *Grep*, and *TeraSort.*

The results show performance improves significantly with the increase in the HDFS block size. This behavior is consistent across all applications when the number of mappers is less than 4. With few mappers, the largest HDFS block size generates an adequate number of map tasks to keep all cores in the system busy. On the other hand, a medium HDFS block size of 256MB and 512MB are preferable for large numbers of cores/mapper slots as it generates more map tasks to run simultaneously with fast execution time per map task. The exception is *Sort,* which is an I/O application. Other applications, including *Wordcount, Grep,* and *TeraSort* show a parabolic behavior at large number of mappers and achieve the minimum execution time at 256MB or 512MB block size. Sort's optimal HDFS block size is 1024MB whereas for *Wordcount* this is 256MB with the maximum number of mappers. Similar to [3], we have observed that *TeraSort* shows hybrid characteristics. The Map phase of *TeraSort* is CPU-bound and Reduce phase is I/O-bound, therefore unlike *Sort, TeraSort's* optimal HDFS block size is 512MB. *Grep* also displays hybrid characteristics. Grep consists of two separate phases, search and sort, running in sequence. The search phase is compute-bound and the sort phase is I/O-bound.

In addition, we have studied the impact of CPU frequencies on performance to understand how MapReduce applications are sensitive to this tuning parameter. The results show that the Sort application is less sensitive to frequency, compared to other benchmarks. For this benchmark when CPU frequency is reduced to half the performance only drops by 9%. Sort is an I/O bound benchmark, which spends most of its execution time requesting data and waiting for I/O operations to complete.

Observation: Although the optimal HDFS block size carefully decided by the application type for the peak performance, using 256MB block size for compute bound and 1024MB for I/O-bound applications can avoid the extensive experimental search to determine the best HDFS block size and achieve close to the upper bound performance.

4.2 Energy-efficiency Analysis

In order to characterize the energy efficiency, we evaluate Energy Delay Product (EDP) metric to investigate trade-off between power and performance. Energy Delay Product (EDP) is a fair metric to study the impact of changing optimization knobs in an architecture. EDP (or Power x ExecutionTime2) represents a trade-off between power and performance. Without EDP and just using energy metric for comparison, we can simply reduce the voltage and frequency in an architecture, and reduce its energy, however at a cost of lowering the performance (increased execution time). Therefore, performance along with energy is important to find out the impact of optimization parameters. The results presented in Figure 2 show that setting the number of mappers equal to the number of available cores minimizes the EDP. The worst EDP is reported with one mapper, while 8 mappers give the best EDP by effectively utilizing all available cores. The margin of EDP improvement becomes smaller with the increase in the number of mappers.

Thus, the best energy efficiency is achieved when we utilize all available cores. In other words, the performance improvement achieved by adding more cores outweighs the power overhead associated with additional cores. The EDP trend is consistent with the execution time trend showing that in I/O bound applications, the maximum energy efficiency is achieved with the largest HDFS block size, however compute-bound and hybrid applications achieve optimal EDP at 256MB and 512MB, respectively. Moreover, we have conducted the analyses of frequency scaling on the EDP results. Energy efficiency is maximized at the highest frequency of 2.4GHz in all applications with the exception of *Sort. Sort* provides the maximum energy efficiency at 1.6GHz. As discussed earlier, *Sort* is an I/O bound application that spends a significant amount of execution time reading data from and writing to HDFS. This makes the performance of *Sort* almost insensitive to the operating frequency.

When we look at the best combination of all these parameters, the results show that by simultaneously fine-tuning the HDFS block size and operating frequency, we can reduce the number of mappers and still be as energy efficient as the maximum number of mappers. For example, Grep with 512 MB block size running at 2.4 GHz frequency with 2 and 4 mappers achieves higher or similar energy efficiency compared to the maximum number of mappers, i.e. 8.

Observation: The results indicate that by fine-tuning frequency and HDFS block size, we can maximize energy efficiency with fewer mappers. Besides, carefully fine-tuning the system and architecture parameter suggests the potential for reducing the reliance on full core occupancy in the system and creates the possibility to co-locate multiple applications onto one node.

4.3 EDP Sensitivity Analysis

To determine how important it is to jointly tune the optimization parameters, we calculate the EDP for various tuning parameters individually and concurrently. If the variation found to be large, it highlights the importance of carefully fine-tuning parameters for energy-efficiency, otherwise an arbitrary

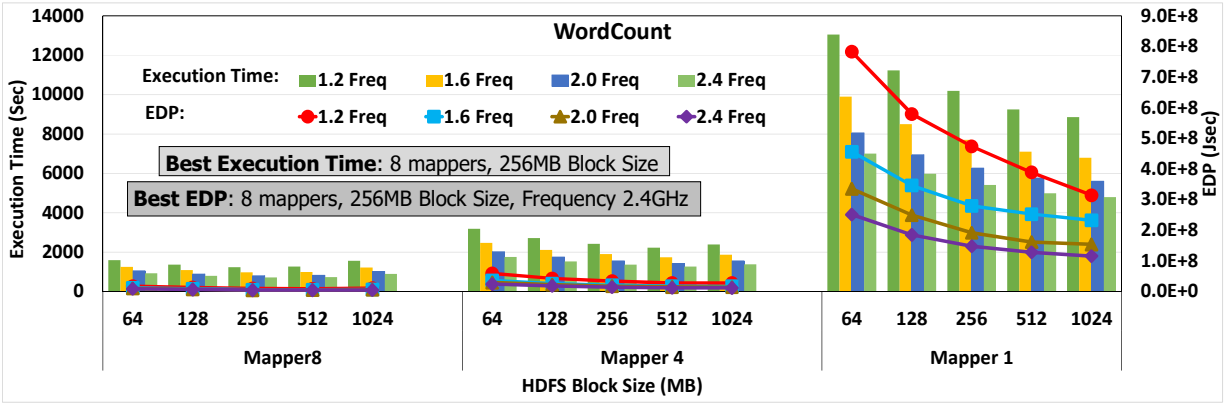

Figure 2(a): Execution Time and EDP of WordCount with various mappers, HDFS block size and operating frequencies

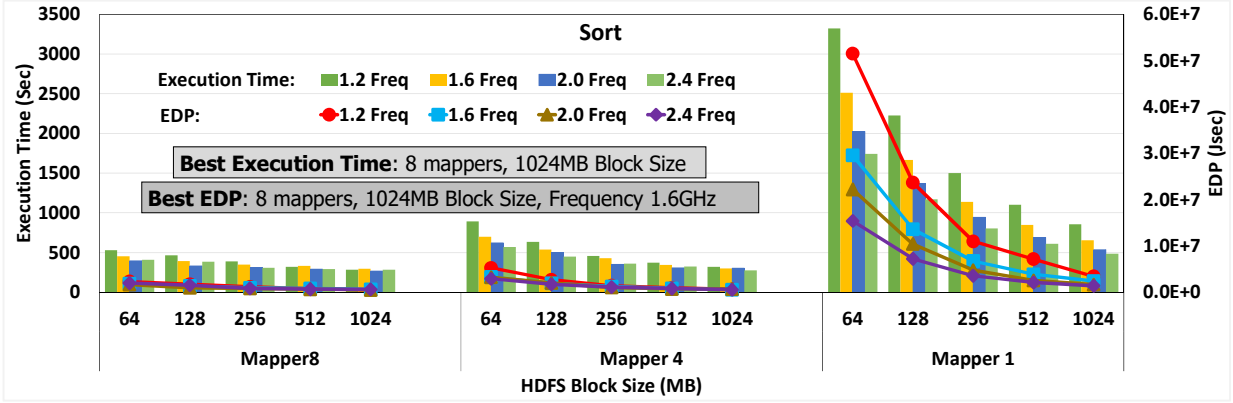

Figure 2(b): Execution Time and EDP of Sort with various mappers, HDFS block size and operating frequencies

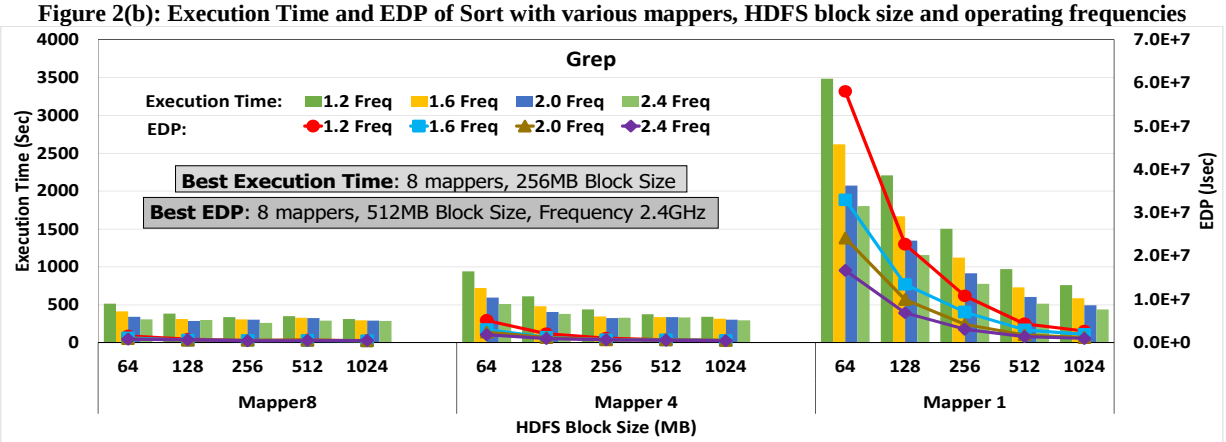

Figure 2(c): Execution Time and EDP of Grep with various mappers, HDFS block size and operating frequencies

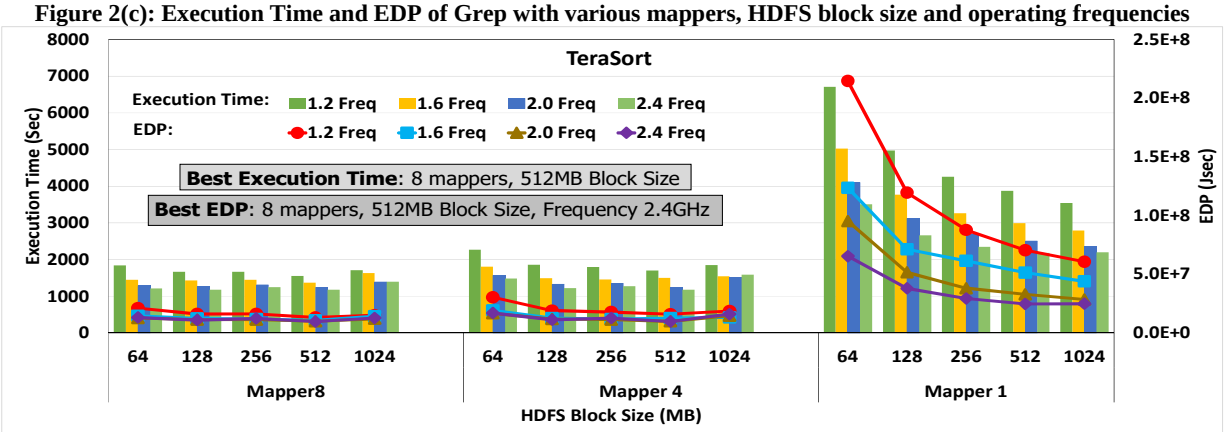

Figure 2 (d): Execution Time and EDP of Terasort with various mappers, HDFS block size and operating frequencies

selection would be sufficient. To understand the variation in energy efficiency with respect to the tuning parameters we present EDP sensitively analysis results in Figure 3(a-d) by changing HDFS block size and frequency individually and concurrently. All EDP results are normalized to the EDP result of 64MB HDFS block size running at the minimum operating frequency of 1.2GHz.

The results show that EDP sensitivity to HDFS block size becomes smaller with the increase in the number of mappers. Similarly, EDP sensitivity to operating frequency becomes smaller with the increase in the number of mappers.

We also observe that the concurrent tuning of HDFS block size and frequency achieves the highest EDP improvement compared to when tuning them individually. The EDP improvement achieved by concurrently tuning HDFS block size and operating frequency ranges from 3.73% to 87.39% compared to the individual tuning parameters. Also the results show that the margin of EDP improvement decreases with the increase in the number of mappers. It is important to note that it is not ideal in a datacenters to assign all cores of a single node to a single application, especially for an I/O intensive application that exhibits a low CPU utilization.

Observation: The results show that applications are more sensitive to frequency and HDFS block size at small number of mappers. Therefore, for co-locating applications on a single node, while each would get fewer mappers/cores allocated, it is critical to determine the fine-tuned these parameters to observe EDP improvement.

5. Co-Locating Applications at the Node Level

The results presented in the previous section show that for MapReduce applications, careful fine tuning of parameters made it more likely that maximum energy efficiency is achieved without utilizing all cores. Thus, we illustrate that co-locating MapReduce applications on the same server are typically effective, particularly when the application types are diverse and have different bottlenecks, as long as they are carefully (and cooperatively) tuned. The alternative is to instead bias toward isolating jobs on servers.

To compare co-located tuned applications with the individually tuned applications, we study two different optimization strategies: individually-located application optimization (ILAO) and co-located application optimization (COLAO). This helps us understand whether tuning MapReduce applications together or individually will provide better EDP.

- ILAO runs the applications serially where each application is tuned individually to achieve the maximum energy-efficiency.
- COLAO runs multiple applications at a node where application tuning parameters are optimized concurrently for maximum energy-efficiency.

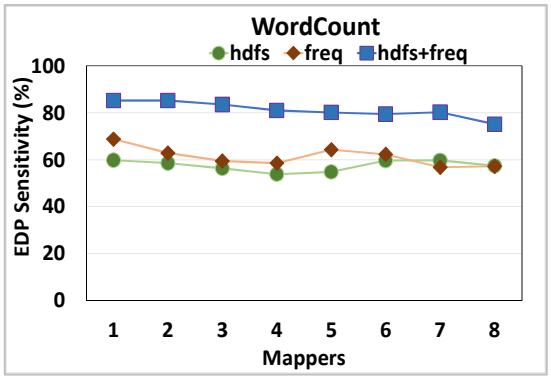

Figure 3(a): WordCount EDP sensitivity analysis w.r.t. HDFS HDFS block size, Frequency (individually) and HDFS block size +Frequency (concurrently)

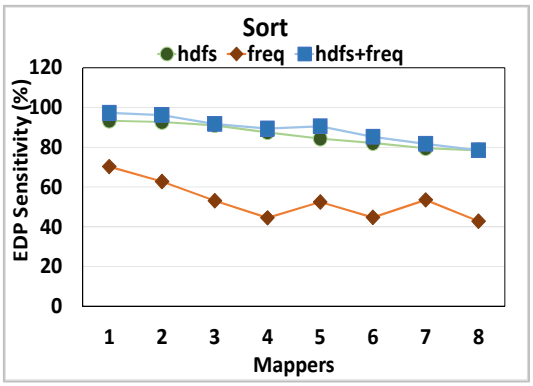

Figure 3(b): Sort EDP sensitivity analysis w.r.t. HDFS block size, Frequency (individually) and HDFS block size +Frequency (concurrently)

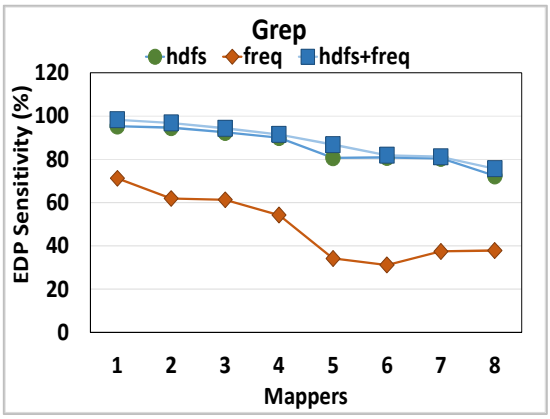

Figure 3(c): Grep EDP sensitivity analysis w.r.t. HDFS HDFS block size, Frequency (individually) and HDFS block size +Frequency (concurrently)

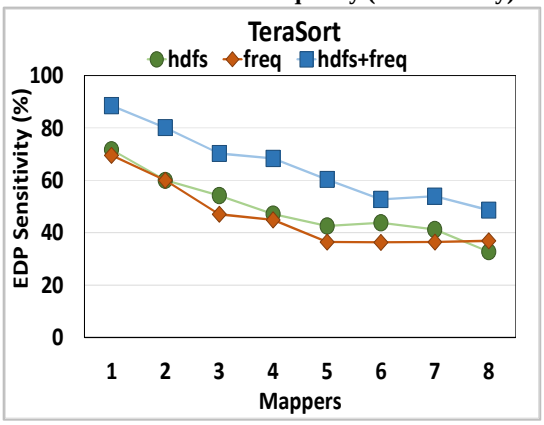

Figure 3(d): TeraSort EDP sensitivity analysis w.r.t. HDFS block size, Frequency (individually) and HDFS block size +Frequency (concurrently)

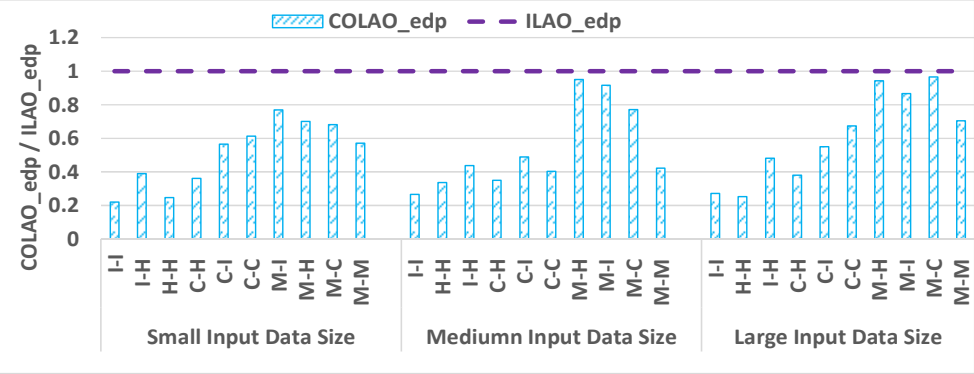

Figure 4: EDP improvement of training workloads with the same input data size

In both studied optimization strategies, various combinations of tuning parameters are explored to find the one that maximizes the energy efficiency. At the node level given the availability of 8 cores we can co-locate 8, 6, 4, 2, and 1 application simultaneously. However, our results indicate that while 2 co-located applications provide improvement over 1 application in terms of energy efficiency, co-locating beyond 2 applications (i.e. 4, 6 and 8) at a node level degrades energy efficiency significantly. Therefore, throughout this paper we focus mainly on co-locating 2 applications at the node level.

Figure 4 presents the EDP ratio of ILAO and COLAO techniques. We have performed experiments with different combinations of input data sizes across all studied applications, however due to space limitation, Figure 4 shows EDP comparison of studied optimization policies when co-located MapReduce applications have same input data size. The presented COLAO results are normalized to their corresponding ILAO values. We observe that in almost all studied cases COLAO outperforms ILAO in terms of EDP (by upto 4.52x). Pairing I/O bound applications together results in the highest EDP gap of 4.52x between COLAO and ILAO. With COLAO optimization technique, co-located applications are running with fewer mappers. For instance, H will run on 5 mappers, and I will be assigned to 3 mappers when H-I applications are co-located on a single node. On the other hand, ILAO will be running each application on the maximum mappers serially. On the other hand, ILAO will be running each application on the maximum mappers serially. On the other hand, the EDP gap reduces between the two techniques when the memory bound applications are co-located with other applications. This is due to the fact that a memory bound application with high execution time typically prefers the maximum number of cores/mappers and suffers when sharing. Overall, the results support the idea of co-locating and concurrent fine-tuning of applications rather than scheduling/fine-tuning them individually.

6. Scalability

In this section, we evaluate the scalability of co-located MapReduce applications on a local cluster with 8 nodes atom servers.

6.1 Application mapping policies

We have evaluated the workloads, shown in Table 1 where each workload comprises of 16 applications. Various application mapping policies are studied with default configuration parameters as well as after tuning configuration parameters. With respect to the number of nodes in a local cluster, the mapping policies studied in this paper are as follows:

1. Serial Mapping [NT]: Each application has access to the entire cluster. [Not Tune-NT] indicates that we are running applications without tuning their configuration parameters. Serial Mapping is referred as SM.
2. Single Node Mapping [NT]: Each application is being assigned to a single node (all 8 cores are active on nodes). Single Node Mapping is referred as SNM.
3. ILAO [T]: Each application is being assigned to a single node and is tuned individually to achieve the maximum energy-efficiency.
4. Core Balance Mapping [NT]: Two applications are co-located on a single node, and the same number of cores (4 cores) is assigned to each application to run. Core Balance Mapping is referred as CBM.
5. COLAO [T]: Co-locate applications at the node level after concurrently tuning the configuration parameters for maximum energy-efficiency.

Figure 5 presents the EDP results for randomly selected workload policies with 8 nodes at the local cluster. All results are normalized to the result of COLAO mapping policy. Serial mapping with no tuning (NT) performs poorly. However, Single Node mapping and ILAO mapping policies that allow multiple applications to run in parallel improve EDP. Furthermore, we have studied the impact of co-locating applications at the node level – Core Balance Mapping and COLAO mapping. Core Balance mapping is sensitive to the behavior of applications in a workload. Compute-bound (C) and memory-bound (M) workloads illustrate poor EDP for Core Balance mapping policy compared to Single Node mapping in the workload WS4, WS5, WS7 and WS8. This is due to the fact that Compute-bound (C) and memory-bound (M) workloads applications with high execution time typically prefers the maximum number of cores/mappers and suffers significant performance loss when sharing.

Additionally, we have observed significant EDP improvement by fine-tuning the configuration parameters of applications as compared to the applications that run without tuning the studied parameters. For instance, COLAO has on average 68.53% and 64.162% better energy efficiency as

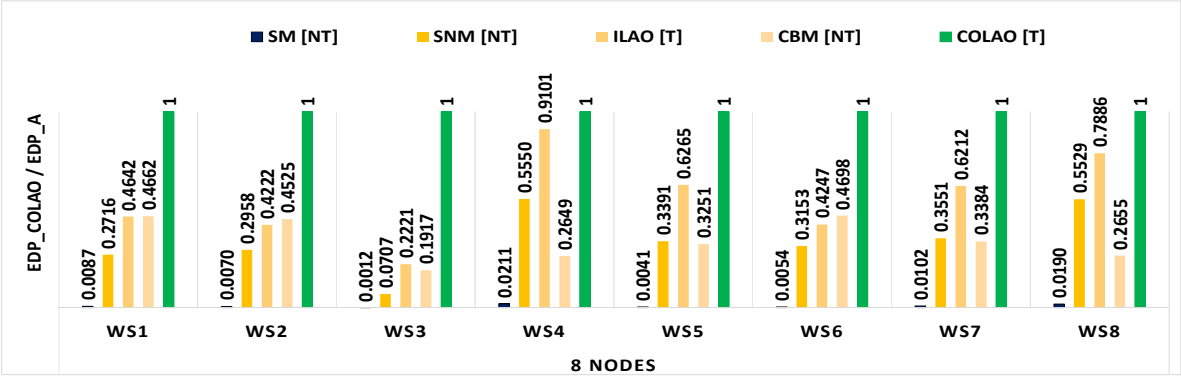

Figure 5: EDP improvement with respect to various mapping scenarios at 8 Nodes
(A = SM, SNM, ILAO, CBM, COLAO)

compared to mapping policies with no tuning i.e. Single Node and Core Balance, respectively. In addition, in the comparison of tuned mapping policy- ILAO and COLAO, we observe that COLAO achieves on average 44.272% better energy efficiency compared to ILAO.

Observation: The results indicate that COLAO mapping policy not only achieve maximum energy efficiency compared to other studied mapping policies especially ILAO, it also reduces the number of processing cores/nodes required to maximize EDP, by fine-tuning configuration parameters and co-locating MapReduce application at a node level.

6.2 Case Study

To validate that COLAO requires less number of nodes to process the MapReduce applications and is still more energy efficient compared to ILAO, the workloads of 16 applications shown in Table 1 are scheduled using ILAO and COLAO mapping policy on a local cluster of 4 and 8 nodes in Figure 6 (a, b). In case of 8 nodes, ILAO can only map 8 applications at a time. On the other hand, COLAO can map all 16 applications simultaneously on 8 nodes by fine-tuning the system, architectural and application configuration parameters.

Figure 6 (a) shows the EDP results for COLAO with 2, 3 and 4 nodes normalized to the results of ILAO at 4 nodes (lower is better). ILAO takes 4 iterations to complete the workload, however, COLAO can complete the execution of all applications in the workload in 2 iterations by co-locating applications simultaneously on 4 nodes. Similar to Figure 4 and Figure 5, EDP results of COLAO_n4 outperforms ILAO_n4. Although, COLAO attains higher EDP at 2 nodes (COLAO_n2) compared to ILAO_n4, the important observation is that the EDP results of COLAO_n3 are comparable to ILAO_n4 for most of studied workloads.

Similarly, in Figure 6(b) where we have considered 8 nodes to execute 16 applications, COLAO outperforms ILAO by 2.24 times on average by accommodating double applications at node level compared to the ILAO. Most importantly, the EDP results for COLAO are also promising with 6 and 7 nodes by the factor of 1.04 and 1.35 times on average compared to ILAO. Compare to the C-bound and M-bound application, I-bound and hybrid applications can still be energy efficiency with less than half number of nodes required by ILAO.

We conclude that despite the increasing complexity of the parameters, through fine-tuning configuration parameters and concurrently running and tuning multiple MapReduce applications, COLAO reduces the number of nodes/cores to execute the applications and even enhances the EDP.

7. Related Work

There has been a significant amount of work to address the challenge of co-locating applications on multicore processor [19, 20]. Several techniques have been developed that perform job scheduling to alleviate the shared resource contention. The work in [7] have introduced a synthetically generated base vectors and have classified the application's usage with respect to the shared resources by co-locating them along the base vectors for selecting the optimal pairing. Additionally, authors have calculated the application's sensitivity; i.e. how

Table 1: Studied workload scenarios

Workload Scenarios	Application type	Studied Applications
WS1	[C,C,C,C,C,C,C,C,C,C,C,C,C,C,C,C]	[svm, svm, wc,wc, svm, wc, mar, wc, mar, mar, wc,wc, mar, wc, svm, wc]
WS2	[H,H,H,H,H,H,H,H,H,H,H,H,H,H,H,H]	[ts, gp, ts, ts, ts, gp, ts, ts, ts, gp, ts, ts, ts, gp, ts, ts]
WS3	[I,I,I,I,I,I,I,I,I,I,I,I,I,I,I,I]	[st, st, st, st, st, st, st, st, st, st, st, st, st, st, st, st]
WS4	[C,C,H,I,C,C,H,I,C,C,H,I,C,C,H,I]	[svm, wc, ts, st, wc, wc, ts, st, mar, svm, ts, st, wc, wc, ts, st]
WS5	[M,H,I,H,M,H,I,H,M,H,I,H,M,H,I,H]	cf, ts, st, ts, cf, ts, st, ts, fp, ts, st, ts, fp, ts, st, ts]
WS6	[H,I,H,I,H,H,I,I,H,I,H,I,H,I,H,I]	[ts, st, ts, st, ts, ts, st, st,ts, st, ts, st,ts, st, ts, st]
WS7	[M,M,M,I,M,M,M,I,M,M,M,I,M,M,M,I]	[cf, cf, cf, st, cf, cf, cf, st, cf, cf, cf, st, cf, cf, cf, st]
WS8	[M,M,H,I,M,M,H,I,C,C,H,I,C,C,H,I]	[cf,fp, ts, st, cf, fp, ts, st, mar, svm, ts, st, wc, wc, ts, st]

performance impacts by lack of a specific processor resource, and application's intensity; i.e. how much application stresses a particular processor resource. Many co-locating studies on CMP platforms [8, 9, 10, 11 and 23] investigate shared cache contention-aware scheduling techniques to improve the performance and fairness. [12] proposed CRUISE that examines the LLC utilization information to schedule multi-programmed applications on CMP. The cache aware scheduler comprises the classification scheme and scheduling policy. Classification scheme is used to identify which co-located applications are effective to schedule and scheduling policy allocates the assigned thread to the cores based on the classification [13]. In [21], authors have used L2 cache miss rate predictions to assign suitable threads together on a CMP platform. In [22], authors model resource interference of server consolidation workloads by estimating cache usage while co-locating two jobs at a time. Bubble-up [30] and Bubble-Flux [41], a characterization and profiling methodology, predicts the performance degradation between pairwise application co-locations. However, they have not discussed how the interplay of tuning parameters impacts the performance and energy efficiency of multiple scheduled applications.

There are also several works that attempt to find which applications should co-locate simultaneously on a CMP. [14] introduces a resource-aware co-locating technique that uses a holistic approach to co-locate the applications for performance and energy improvement. The effectiveness of this approach is dependent on the studied multi-programmed workload that comprises of a mixture of high contention and low contention applications. The work in [15] has studied the co-located HPC applications by evaluating the affinity-aware contention information with the greedy allocation heuristics technique. Our work is orthogonal to these resource-awareness techniques.

Big data frameworks and in particular Hadoop-based applications [16, 26, 27] inherent different microarchitectural behavior than traditional application (SPEC and PARSEC) [1, 28, 29, 32, 33, 34]. In addition, these frameworks have large set of tuning knobs, which individually and concurrently influence the mapping decision. All of above techniques therefore are not directly applicable for co-locating outcome of MapReduce applications. It is also important to note that most of prior research that focus on scheduling has shown promising results, however using simulation-based methods [37], which cannot capture the real-system behavior of complex big data framework.

While several recent work show [42, 43] how tuning individual or a subgroup of tuning parameters at a time improves performance, they have ignored the interplay among all of these parameters at various level of abstractions. [42, 43] mainly focused on fine-tuning optimization parameters for individual applications and in isolation, however, our work targets the opportunities for co-optimizing these tuning parameters for multiple scheduled applications, simultaneously. [44, 45] use online classification to estimate interference between co-located workloads that are unlikely to cause interference, however, [44] does not study the impact of tuning parameters. These studies focus on performance analysis as compared to our work that emphasizes on energy efficiency. Furthermore, unlike [43] and

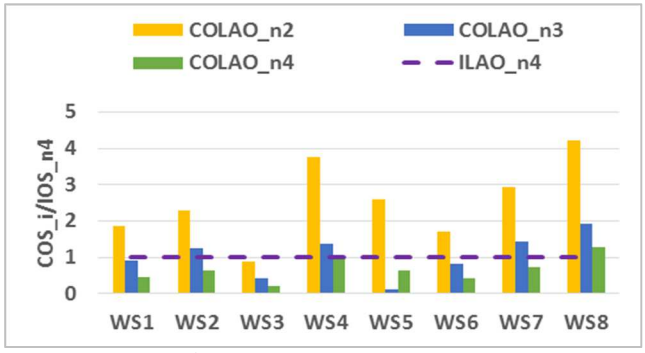

Figure 6 (a): EDP analysis of COLAO and ILAO at 4 Nodes (i = Nodes equal to 2, 3, 4)

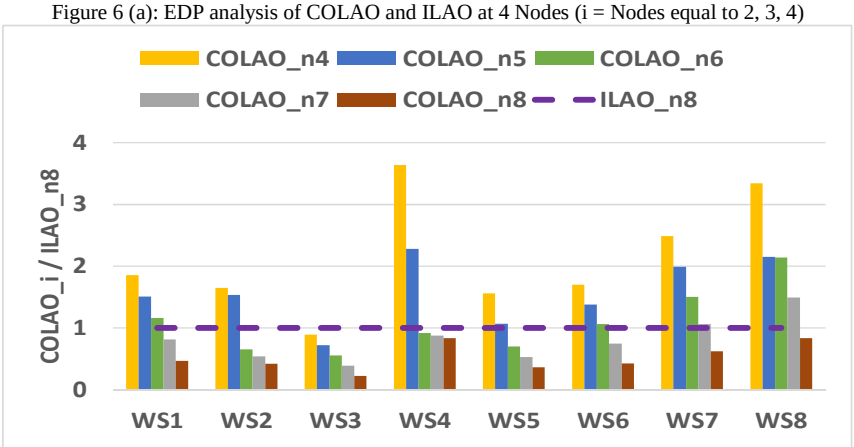

Figure 6 (b): EDP analysis of COLAO and ILAO at 8 Nodes (i = Nodes equal to 4, 5, 6, 7, 8)

[45], our results illustrates that HDFS block size has a significant impact on the performance and energy efficiency. In addition, [43] has performed a limited study and on only two small Hadoop kernels.

8. Conclusions

MapReduce applications with their complex and deep software stacks, is influenced by many tuning parameters such as number of mappers, HDFS block size, and frequency of the core. The large number of tuning parameters provides more opportunity for optimization, but it is also challenging problem.

This paper examines the impact of tuning parameters and the interplay among them on performance and energy efficiency. We observe that although maximum energy efficiency on a single node is achieved while utilizing all available cores/mappers slots, the reliance on the maximum number of cores reduces significantly after concurrently fine-tuning parameters such as frequency and HDFS block size. This provides opportunities to co-locate multiple MapReduce applications at the node level. In addition, the level of sensitivity of EDP to these parameters when running applications with fewer mapper slots/cores increases significantly, highlighting the importance of fine-tuning when co-locating multiple applications onto one node. Comparing two scheduling strategies where one fine-tune applications individually and run them serially and the other where fine-tune applications together and co-locate them at the node level, show that concurrent fine-tuning of MapReduce applications (at application, architecture and system levels), and co-locating them reduces the number of nodes required to execute MapReduce applications by half while improving the EDP by 2.2X, on average, for a wide range of studied workloads.

References

[1] Malik, M. et al. “Big data on low power cores: Are low power embedded processors a good fit for the big data workloads?” in ICCD 2015.

[2] Ferdman, M. et al. "Clearing the clouds: a study of emerging scale-out workloads on modern hardware." ACM SIGPLAN 2012.

[3] Huang, S., et al. "The HiBench benchmark suite: Characterization of the MapReduce-based data analysis," In the proc. of 26th ICDEW, 2010

[4] “Perf,” https://perf.wiki.kernel.org/index.php/Main Page.

[5] “Wattsuppro power meter,” [online]

[6] “Dsat,” http://lintut.com/dstat-linux-monitoring-tools/.

[7] Doucette, D., et al., “Base vectors: A potential technique for microarchitectural classification of applications,” in WIOSCA 2007.

[8] Jiang, Y., et al., “Analysis and approximation of optimal co-scheduling on chip multiprocessors,” in PACT 2008

[9] Xie, Y., et al., “Dynamic classification of program memory behaviors in cmps,” in the 2nd CMP-MSI 2008.

[10] Knauerhase, R., et al., “Using os observations to improve performance in multicore systems,” in IEEE MICRO 2008.

[11] Chandra, D., et al., “Predicting inter-thread cache contention on a chip multi-processor architecture,” in HPCA 2005

[12] Jaleel, A., et al., “Cruise: cache replacement and utility-aware scheduling,” in ACM SIGARCH Computer Architecture News, 2012

[13] Blanche, A. D., et al., “Addressing characterization methods for memory contention aware co-scheduling,”Journal of SC, vol.71,1451–1483, 2015.

[14] Bhadauria, M., et al., “An approach to resource-aware co-scheduling for cmps,” in ACM ICS 2010

[15] Kim, S., “Plat-form and co-runner affinities for many-task applications in distributed computing platforms,” in CCGrid 2015

[16] Xiong, W., et al., “A characterization of big data benchmarks,” in Big Data, 2013

[17] Jia, Z., et al., “Characterizing and subsetting big data workloads,” in IISCW 2014

[18] Apache Mahout: scalable machine-learning and data-mining library

[19] Hankendi, C. et al., "Energy-efficient server consolidation for multi-threaded applications in the cloud." In IGCC 2013

[20] Dey, T. et al., "Characterizing multi-threaded applications based on shared-resource contention." In ISPASS 2011

[21] Fedorova, A. et al., "Performance of multithreaded chip multiprocessors and implications for operating system design." (2005).

[22] Tang, L. et al., "The impact of memory subsystem resource sharing on datacenter applications." In ISCA 2011

[23] Kim, Y. K., et al., "A symbiotic evolutionary algorithm for the integration of process planning and job shop scheduling." Computers & Operations Research 2003

[24] Che, S., et al., “A characterization of the Rodinia benchmark suite with comparison to contemporary CMP workloads.” In IISWC 2010

[25] Malik, M., et al. "System and architecture level characterization of big data applications on big and little core server architectures." In Big Data 2015

[26] Li, A., et al. "CloudCmp: comparing public cloud providers." Proc. of the 10th ACM SIGCOMM conf. on Internet measurement. ACM, 2010

[27] Armstrong, T.G., et al., “LinkBench: a database benchmark based on the Facebook social graph.” In Proc. of ACM SIGMOD 2013

[28] Anwar, Ayesha, et al., "On the use of microservers in supporting hadoop applications." In CLUSTER, 2014

[29] Krish, K. R., et. al., "[phi] Sched: A Heterogeneity-Aware Hadoop Workflow Scheduler." In MASCOTS, 2014.

[30] Mars, J., et al. "Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations." In MICRO 2011.

[31] Bienia, C. et al., “Benchmarking modern multiprocessors. “ New York: Princeton University 2011

[32] Chung, E. S., et al., “Linqits: Big data on little clients.” In ACM SIGARCH Computer Architecture News 2013

[33] Loghin, D., et al., “A performance study of big data on small nodes. Proceedings of the VLDB Endowment, 8(7):762–773, 2015.

[34] Wang, L. et al., “Bigdatabench: A big data benchmark suite from internet services.” In HPCA 2014

[35] Aktasoglu, M. S. "A Workload Mapping Method for Multicore Systems Using Cross-run Statistics." PhD diss., The Penn State University, 2012.

[36] Blem, E. et al., "Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures." In HPCA2013

[37] Liu, Y., et al., “SleepScale: Runtime Joint Speed Scaling and Sleep States Management for Power Efficient Data Centers,” in ISCA 2014

[38] Wu, C. M., et al., "A green energy-efficient scheduling algorithm using the DVFS technique for cloud datacenters." Future Generation Computer Systems 37 (2014): 141-147.

[39] Holmes, G., et al., “Generating rule sets from model trees”, Sydney, Australia: Springer-Verlag, 1999.

[40] Zhao. Y., et al., "Comparison of decision tree methods for finding active objects," Advances in Space Research, vol. 41, pp. 1955-1959, 2008.

[41] Yang, H., et al. "Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers." ACM SIGARCH Computer Architecture News 2013.

[42] Ganapathi, A., "Predicting and optimizing system utilization and performance via statistical machine learning." 2009.

[43] Yigitbasi, N. et al., "Towards machine learning-based auto-tuning of mapreduce." In MASCOTS 2013.

[44] Delimitrou, C. et al., "Paragon: QoS-aware scheduling for heterogeneous datacenters." In ACM SIGPLAN Notices 2013.

[45] Delimitrou, C. et al., "Quasar: resource-efficient and QoS-aware cluster management." In ACM SIGPLAN Notices 2014.

[46] http://www.storageswitzerland.com/Articles/Entries/2013/5/20_What_Is_A_Hyperscale_Data_Center.html

[47] Caulfield, A. M., et al., “A cloud-scale acceleration architecture.”, In MICRO 2016

[48] Malik, M. et al., “Big vs Little Core for Energy-Efficient Hadoop Computing”, Design, Automation and Test in Europe (DATE), 2016

[49] Neshatpour, K., et., al., “Accelerating Machine Learning Kernel in Hadoop Using FPGAs”, ,in CCGRID, 2015