ECoST: Energy-Efficient Co-Locating and Self-Tuning MapReduce Applications

Maria Malik[1], Hassan Ghasemzadeh[2], Tinoosh Mohsenin[3], Rosario Cammarota[4], Liang Zhao[1], Avesta Sasan[1], Houman Homayoun[5,] Setareh Rafatirad[1]
[1]George Mason University, Fairfax, VA, USA, {mmalik9, lzhao9, asasan, srafatir}@gmu.edu
[2]Department of Computer Science, Washington State University, USA, hassan@eecs.wsu.edu
[3]Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County,tinoosh@umbc.edu
[4]Intel, USA, rosario.cammarota@intel.com
[5]University of California Davis, CA, USA

ABSTRACT

Datacenters provide high performance and flexibility for users and cost efficiency for operators. Hyperscale datacenters are harnessing massively scalable computer resources for large-scale data analysis. However, cloud/datacenter infrastructure does not scale as fast as the input data volume and computational requirements of big data and analytics technologies. Thus, more applications need to share CPU at the node level that could have large impact on performance and operational cost. To address this challenge, in this paper we show that, concurrently fine-tune parameters at the application, microarchitecture, and system levels are creating opportunities to co-locate applications at the node level and improve energy-efficiency of the server while maintaining performance. Co-locating and self-tuning of unknown applications are challenging problems, especially when co-locating multiple big data applications concurrently with many tuning knobs, potentially requiring exhaustive brute-force search to find the right settings. This research challenge upsurges an imminent need to develop a technique that co-locates applications at a node level and predict the optimal system, architecture and application level configure parameters to achieve the maximum energy efficiency. It promotes the scale-down of computational nodes by presenting the Energy-Efficient Co-Locating and Self-Tuning (ECoST) technique for data intensive applications. ECoST proof of concept was successfully tested on MapReduce platform. ECoST can also be deployed on other data-intensive frameworks where there are several parameters for power and performance tuning optimizations. ECoST collects run-time hardware performance counter data and implements various machine learning models from as simple as a lookup table or decision tree based to as complex as neural network based to predict the energy-efficiency of co-located applications. Experimental data show energy efficiency is achieved within 4% of the upper bound results when co-locating multiple applications at a node level. ECoST is also scalable, being within 8% of upper bound on an 8-node server.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
ICPP 2019, August 5-8, 2019 Kyoto, Japan
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6295-5/19/08..$15.00
https://doi.org/10.1145/3337821.3337834

1. INTRODUCTION

Today's data centers are playing a critical role to process diverse applications in the emerging domain of big data. Hyperscale datacenters have gained interest as a promising computing architecture that is designed to provide a scalable solution to process mass volume of data [42]. Recent improvements in the networking, storage, processing and infrastructure management [10, 11] 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 total cost of ownership (TCO) in hyperscale data centers is one of the major limiting scaling factors. To address these challenges, many recent works address the need to use hardware specialized accelerators to increase the performance by using fewer number of nodes [11]. 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 increase in heterogeneity and in the demand of data center workloads causes an increase in the infrastructure operating costs. With the substantial expected increase in the operational cost reflected by the higher energy consumption and cooling cost of the data center, a need for microserver-based architectures has been proposed as an alternative to traditional high-performance architectures to process big data applications [1, 23, 41].

With the significant increase in the volume of data, more applications are migrating to cloud. However, cloud infrastructure is not scaling as fast as the size of data is increasing. Thus, more applications need to be scheduled at the node level. Therefore, the question of how to co-locate and configure data-intensive applications automatically for energy efficiency becomes important to contain the TCO. In a data-intensive framework such as MapReduce, configuration parameters, as well as application and architectural parameters, directly affect its performance and energy efficiency [22]. Through an extensive and methodical investigation of performance and energy-efficiency of MapReduce applications, [32] 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. The results 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. In addition, authors have discussed two optimization strategies; Individually-Located Application optimization (ILAO) that runs the applications serially where each application is tuned individually and Co-Located Application Optimization (COLAO) that tunes the collocated applications concurrently at a node level. Given that both of these techniques are offline and rely on extensive brute force search to find the best configuration parameters for co-located MapReduce applications, there is an imminent need to develop a technique that can identify which applications to co-locate at a node level and how to self-tune the optimal configure parameters at the system, architecture and application level to achieve the maximum energy efficiency. Towards this goal, we develop an energy efficient co-locating and self-tuning technique, referred as ECoST, for data intensive applications. While ECoST proof of concept has been successfully tested on MapReduce framework, which is still a de facto programming standard of big data and has been adapted widely by industry and has a large community [2, 3], it can also be deployed in other data-intensive frameworks where there are several parameters for power and performance tuning optimizations.

The key features of ECoST are 1) pairing applications to co-locate at a single node level with the minimum performance degradation due to resource sharing, and 2) self-tuning prediction (STP) techniques that predict the optimal application, system and architectural parameters at run-time for the co-located MapReduce applications to achieve maximum energy efficiency. By decoupling the decision to co-locate applications and tuning parameters concurrently, we significantly simplify the complex optimization problem of searching for the best pair of applications to co-locate at the node level, and simultaneously navigating the wide array of tuning parameters for each application. In addition, we develop distinct machine learning based models to implement STP including Linear Regression (LR), REPTree, and Multilayer Perceptron (MLP) as well as the Lookup Table (LkT) model. These learning models are significantly different as they have different complexity. All of these predictive methods are online and will be compared with two offline scheduling strategies ILAO and COLAO mention in [32].

A closest work to ours is Bubble-up [20], in which authors 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 [17]) has been well studied in the literature [8, 12]. However, data intensive applications such as in MapReduce, have fundamentally different micro-architectural behavior than traditional applications highlighted in recent work [23], while at the same time 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 [16, 14] 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, which is a more challenging problem to address.

To the best of our knowledge, this is the first work that addresses these challenges to identify which applications to co-locate at a node level and determine machine-learning based self-tuning predictive models to maximize the energy efficiency. In this paper, our analysis helps to determine how critical it is to jointly fine tune system, application and architecture level parameters that creates new opportunities to co-locate MapReduce applications concurrently, and how to determine these best tuning parameters at the system, application, and architecture using self-tuning prediction techniques to achieve the maximum energy efficiency.

2. EXPERIMENTAL SETUP

This section describes our hardware and software platforms used to run real experiments on reasonable server hardware, studied application 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. It is important to note that while the choice of microserver for self-tuning is more challenging due to limited availability of hardware performance counters compared to high performance servers such as Xeon, all of the methodologies and solutions presented in this work can be applied to high performance servers as well and similar results can be observed. Considering that MapReduce applications are sensitive to the studied configuration parameters regardless of the CPU architecture type, optimization strategies results would remain valid on a high-performance server architectures as well.

Each studied Intel Atom has 8 processor cores per node and a two-level cache hierarchy with 8GB of system memory using DDR3 @1600MHz. Although, new big data frameworks have emerged in recent years in response to continues changes in big data analytics field; examples include Spark, Tez, Flink, and Petuum, just to name a few, MapReduce framework such as Hadoop and Spark are still a de facto standard of big data which has been adapted widely by industry and has a large community. While not all emerging big data analytics framework can certainly be included for characterization and benchmarking in one research work, this paper focuses on the fundamental Hadoop MapReduce framework for a proof of concept of ECoST. Our solutions can also be applied to Spark MapReduce applications, as well as other data intensive frameworks that have many power and performance tuning knobs. For this study, we have selected the parameters that are system configurable and transparent to the user space, 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 affect 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. Four applications are Hadoop micro-benchmarks, namely Wordcount-WC, Sort-ST, Grep-GP and TeraSort-TS that are used as kernels in many big data applications.

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 K-Mean (KM) [19].

The studied big data workloads are representative programs from 15 different domains such as graph mining, data mining, data analysis platform and pattern searching applications, which are frequently used in real world applications. Several recent work have included these selected applications for benchmarking and characterization of Hadoop-based big data applications [18].

2.3 Input Data Size

The size of data can have significant impact on micro-architectural behavior [22]. For this research, we therefore use 3 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. While MapReduce was originally introduced to process multi-terabyte data in scale-out clusters, most MapReduce workloads have a footprint in the GB range with the median of 14GB [30]. Hadoop exploits cluster-level infrastructure with many nodes for processing big data applications, however, the experimental data should be collected at the node level to understand how various optimizations and scheduling decisions affects the performance, architectural parameters and energy-efficiency at the node level.

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 Tools

We use Perf to capture the performance characteristics of the studied applications. Perf multiplexes the Performance Monitoring Unit (PMU) to measure performance as well as other hardware events, therefore, to obtain accurate values for several hardware events, we run each workload multiple times. For measuring power consumption, Wattsup PRO power meter [5] 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 power dissipation of the core. The same methodology is used in [22], for power and energy analysis. We use Dstat [6], a system-monitoring tool for main memory, disk, and CPU utilization analysis and Weka toolkit [4] to build our machine learning based predictive models.

2.6 Energy Efficiency Metric

In order to characterize the energy efficiency analysis, we evaluate Energy Delay Product (EDP) metric to investigate trade-off between power and performance [31]. Energy Delay Product (ExecutionTime x ExecutionTime x Power) is a fair metric to study the impact of changing optimization knobs of an architecture. 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.

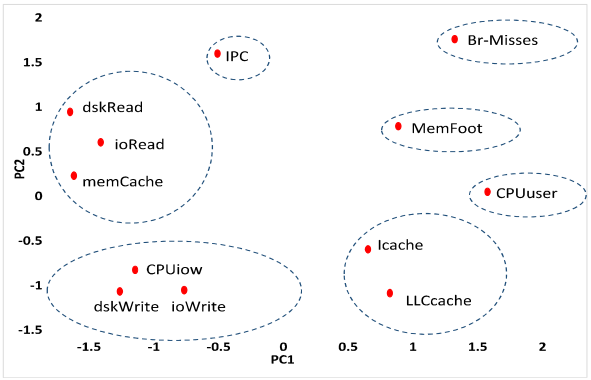

Figure 1: Scatter plot of feature metrics using first and second principal components

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 [1, 29]. 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

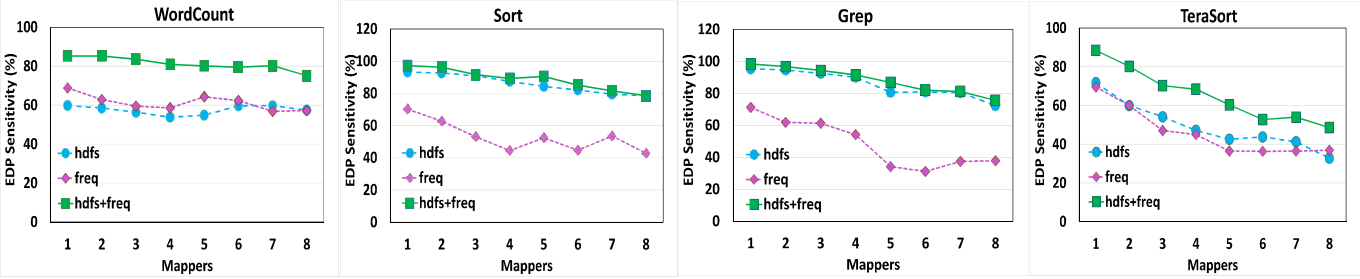

Figure 2: EDP improvement analysis w.r.t. HDFS block size and Frequency (individually) and HDFS block size + Frequency (concurrently)

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. In Figure 1, 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.

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, I/O-bound referred to as hybrid (H), memory-bound (M), and I/O-bound (I) classes. For instance, the CPU user utilization of wordcount is higher than the average user utilization of the studied applications and with low CPU iowait utilization and I/O bandwidth (read/write) rates this application is categorized as compute intensive. We observe 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 micro-architectural behavior. The same methodology is used in [34].

4. Motivation for Fine Tuning and Co-locating Applications at Node Level

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 us to determine, 1) whether fine tuning parameters create new opportunities for co-scheduling applications at the node level, and 2) whether it is important to carefully fine-tune parameters to co-locate applications at the node level and still be energy efficient.

4.1 EDP Improvement 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 EDP improvement is found to be large, it highlights the importance of carefully fine-tuning parameters for energy-efficiency; otherwise, an arbitrary selection would be sufficient. To understand the variation in energy efficiency with respect to the tuning parameters we present EDP improvement analysis results in Figure 2 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 improvement to HDFS block size becomes smaller with the increase in the number of mappers. Similarly, EDP improvement 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. In addition, 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 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.

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

4.2 Co-Locating Applications at the Node Level

As discussed in the previous section, careful fine-tuning of

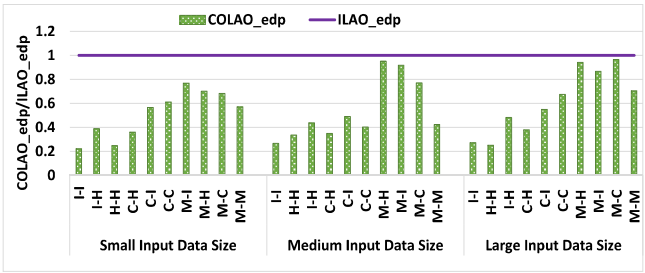

Figure 3: EDP improvement of training workloads with the same input data size

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.

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.

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 3 presents the EDP ratio of ILAO and COLAO techniques. The studied applications are classified 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 have performed experiments with different combinations of input data sizes across all studied applications, however due to space limitation; Figure 3 shows EDP comparison of studied optimization policies when co-located MapReduce applications have same input

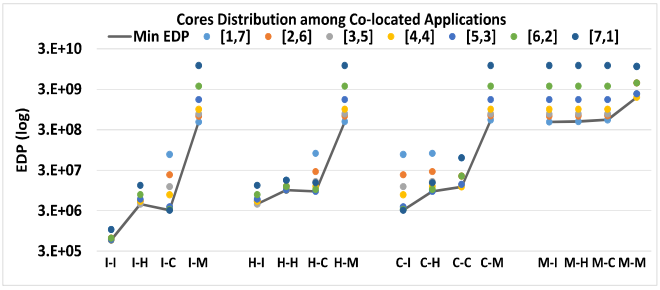

Figure 5: Priority ranking for each co-located class based on EDP

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. However, the EDP gap reduces between the two techniques when the memory bound applications are co-located with other applications. This is because 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.

Given that both of these techniques are offline and rely on extensive brute force search to find the best configuration parameters for co-located MapReduce applications, there is an imminent need to develop a technique that can identify which applications to co-locate at a node level and how to self-tune the optimal configure parameters at the system, architecture and application level to achieve the maximum energy efficiency.

5. Energy-Efficient Co-locating and Self-Tunning (ECoST)

In this section, we propose our technique for energy-efficient

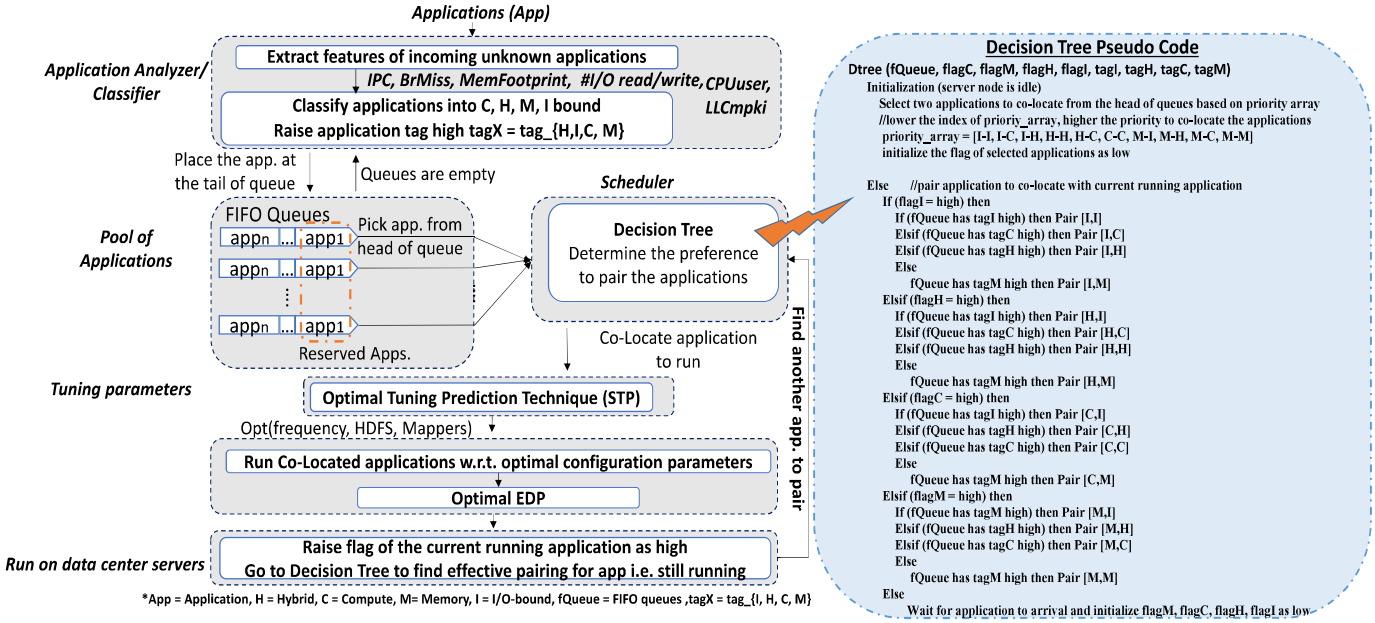

Figure 4: Overview of ECoST

co-locating and self-tuning of MapReduce applications, called ECoST. The purpose of this technique is to first classify the incoming unknown applications to the datacenter in terms of their architectural behavior and then co-locate and fine-tune them on the microserver to achieve maximum energy efficiency. By decoupling the decision to co-locate and tune parameters, ECoST significantly simplifies the complex optimization problem of searching for the best combination of applications to co-locate onto a single node, and simultaneously navigating the wide array of tuning parameters for each application and their interactions between the applications. Figure 4, shows an overall view of the ECoST. New jobs (applications) are arriving to the datacenter and are placed at the tail of a wait queue. The wait queue is implemented as a FIFO where applications are scheduled from the head of the queue. To avoid starvation, the job at the head of the queue has a reservation. A small job is allowed to leap forward as long as it does not delay the job at the head of the queue [24, 40].

Based on what observed in the previous section, it is more energy-efficient to co-locate and concurrently fine-tune applications rather than running/fine-tuning them individually. Therefore, for ECoST scheduler we are assuming that two applications are scheduled (running) on each server node, and several other applications are waiting in the wait queue to be paired as soon as any one of the two applications finishes its execution.

Thus, the research question becomes in steady state which application from the wait queue to pair with the current application running on the server node. To address this issue, ECoST works in the following steps.

Step1: Incoming Application Analyzer/Classifier

The classifier identifies the behavior and characteristics of unknown incoming applications to the datacenter. First, ECoST extracts distinct architectural features from the application. Second, it classifies the application based on the characteristics of known (training) applications. The training applications are classified 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)

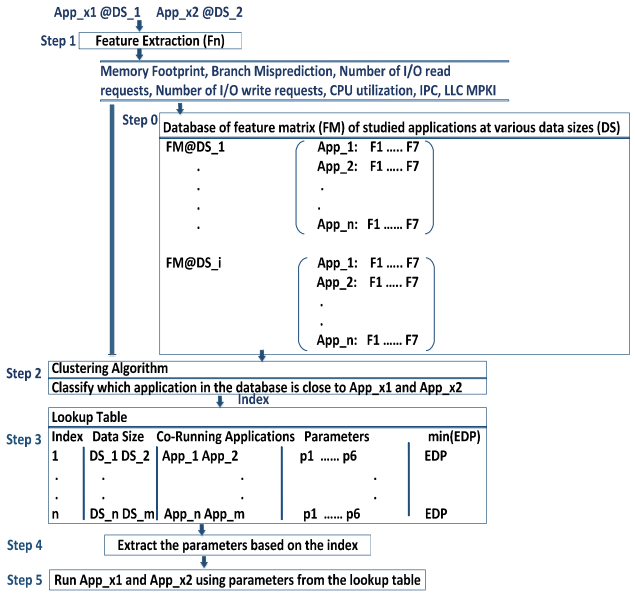

Figure 6: Lookup Table-based Self-tuning prediction technique (LkT-STP)

Table 1: Absolute Percentage Error (%) of training applications

	LR	REPtree	MLP
C-C	38.6	5.67	0.85
C-I	32.49	2.65	0.35
C-H	50.91	4.94	0.86
I-I	51.12	4.87	1.54
I-H	55.44	5.13	1.148
I-M	69.09	2.64	0.11
H-H	52.90	9.64	1.92
H-M	68.92	2.63	0.11
M-M	62.86	2.96	0.66
C-M	69.69	2.69	0.13
Average	55.20	4.38	0.77

classes.

Once an incoming application is classified based on collected architectural features, it is tagged accordingly and steers into the pool of applications queue; wait queue. As discussed the wait queue is implemented as FIFO, where applications are allocated from the tail and selected from the head.

Step 2: Scheduler

The ECoST scheduler attempts to find an application from the head of wait-queue and co-locate it to the node that is currently running another application. The goal of the scheduler is to select the applications to co-locate, which minimizes degradation relative to running alone. This is in fact a local decision and only attempts to maximize the energy-efficiency for the applications already reserved in the head of the queue. It is important to note that the reservation of applications at the head of the queue is important as it can alleviate possible starvation of low priority application [24, 40]. As discussed, a small job is allowed to leap forward as long as it does not delay the job at the head of the queue. To drive the pairing strategy we use application class information. To this goal, we use priority ranking of each class, derived from the data in Figure 5. This figure shows the EDP results for every combination of two applications along with every combination of core partitioning with the tuned configuration parameters. The solid line shows the lowest EDP for each pair of applications across all core partitioning scenarios. We rank each application pair based on the lowest EDP they have, within all combinations of core partitioning. As shown, I-I ranks first and achieves the lowest EDP across all studied cases. I-C, I-H, H-H, H-C, C-C are ranked next. It is interesting to note that for M application, no matter what other application it pairs with, it achieves the highest EDP. Therefore M-X (X being I, H, C or M) receives the lowest ranking.

To find out which application ECoST selects from the wait queue to schedule and run along with the current application running on the server, we analyze the data presented in Figure 5. As shown, I-I has the lowest EDP compared to I-H, I-C, and I-M; i.e. if the current application is I, then co-locating it with another I application waiting in the wait queue minimizes the EDP across all studied cases. Interestingly, when the current application is H, or C, or even M, pairing it with an I application from the wait queue minimizes the EDP. This indicates that, to minimize the EDP, application with I class in the wait queue should be given the highest priority to be co-located with any other application running on the server node. The next priority is given to H or C applications, since based on the results in Figure 5, co-locating them with any other running applications results in the lowest EDP. Finally, the lowest priority is given to M applications.

Based on this offline analysis, we implemented a simple

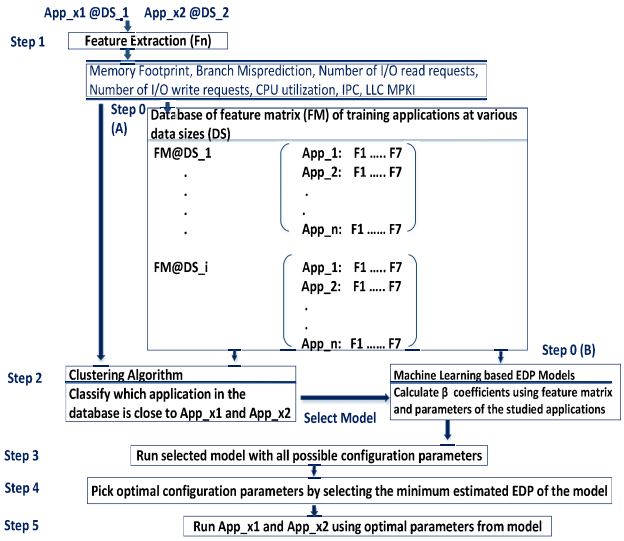

Figure 7: Machine Learning Model-based Self-tuning prediction technique (MLM-STP)

decision tree to find which application to co-locate simultaneously. The decision tree pseudo code is presented in Figure 4.

After pairing, ECoST fine-tunes the architectural, system, and application level parameters of the paired applications concurrently. This is important, as the results in the previous section highlighted the high level of sensitivity of EDP to these parameters particularly when running applications with fewer mappers (which occurs when two or more applications are co-located at the node level and therefore each get fewer mappers). ECoST uses a predictive model referred to as self-tuning prediction techniques, or STP, to estimate the EDP of each application across a large range of tuning parameters. Later in this paper, we provide more details on the predictive models we developed and show how they perform compared to offline oracle techniques that require extensive brute force search to find the best tuning and scheduling.

6. Self-tuning prediction techniques

As results suggested in the previous section, COLAO achieves significant energy efficiency improvement over ILAO across all studied applications, Figure 3. However, unlike ILAO where the tuning parameters for individual application are correlated to its behavior, whether I/O bound, memory bound, compute-bound or hybrid, and can be heuristically set for each class of applications, in COLAO the two applications are competing for the shared resources creating complex interactions between them. Therefore, while brute force search in ILAO can be avoided by simple heuristics based on standalone application behavior, COLAO can require an exhaustive brute-force search to find the optimal tuning parameters over all possible permutations of tuning parameters of the two applications. In this section we present self-tuning prediction techniques (STP), that attempt to find the best application, system, and architectural parameters for incoming unknown MapReduce applications to provide the maximum energy efficiency. STP leverages existing hardware performance counter information to derive the prediction algorithm.

6.1 Feature Metrics Selection

Based on the selected resource utilization features and micro-architectural parameters (Fn), reported in Section 3.2, we construct the feature matrix (FM) for training applications that will help predict the behavior of unknown MapReduce applications (testing applications) based on their resource profiling and micro-architectural characteristics. We implement the code that classifies the application into one of the four classes i.e Compute-bound (C), Hybrid (H), I/O-bound (I) and Memory-bound (M). For instance, the CPU user utilization of wordcount is higher than the average user utilization of the studied applications and with low CPU iowait utilization and I/O bandwidth (read/write) rates this application is categorized as compute intensive.

6.2 Database Construction

To determine what tuning parameters provide the maximum energy efficiency for co-located unknown applications (testing set), we rely on a database which stores the best configuration parameters for a set of known applications (training set). The database is used to predict the best configuration parameters for unknown incoming applications to achieve the maximum energy-efficiency based on application characteristics and input data size, without requiring exhaustive brute-force experimental analysis to test all possible tuning parameter permutations for all co-located applications. Because the database is populated with the best results for various co-located applications, this allows us to navigate a complex set of highly co-dependent parameters, after simply characterizing each application in isolation.

6.3 Machine Learning based EDP Models

We have selected three machine learning-based models for predicting optimal configuration of co-located applications. These models include linear regression (LR), non-linear regression decision tree (REPTree), and Multilayer Perceptron (MLP) which is an artificial neural network model. The main reason for selecting these models is that they represent three different types of learning classifiers; i.e. regression, decision tree and neural network emphasizing different accuracy and complexity level. Each model

Table 2: Error estimation among COLAO, LkT, LR, MLP and REPTree techniques for subset of studied applications

Co-Located Apps.	Configurations (Freq_app1, hdfs_app1, map_app1 --- Freq_app2, hdfs_app2, map_app2)																												Error (%) wrt COLAO (Oracle)			
	COLAO (Oracle)						LkT						LR						MLP						REPTree									
	f1	h1	m1	f2	h2	m2	f1	h1	m1	f2	h2	m2	f1	h1	m1	f2	h2	m2	f1	h1	m1	f2	h2	m2	f1	h1	m1	f2	h2	m2	LkT	LR	MLP	REPTree
H-H	2.4	1024	3	2.4	512	5	2.4	512	3	2.4	512	5	2.4	512	3	2.4	512	5	2.4	512	3	2.4	512	5	2.4	512	3	2.4	512	5	0.40	0.40	0.40	0.40
C-M	2.4	1024	1	2.4	512	7	2.4	1024	1	2.4	512	7	1.2	256	3	2.4	128	5	2.4	1024	1	2.4	256	7	2.4	1024	1	2.4	256	7	3.89	37.26	3.06	3.15
I-M	2.4	1024	1	2.4	512	7	2.4	512	1	2.4	256	7	2.4	1024	1	2.4	256	7	2.4	1024	1	2.4	256	7	2.4	1024	1	2.4	256	7	4.78	3.67	3.67	3.77
H-M	2.4	512	1	2.4	512	7	2.4	512	1	2.4	256	7	2.4	512	4	2.4	512	4	2.4	512	1	2.4	256	7	2.4	512	1	2.4	256	7	3.43	36.83	3.43	3.43
I-H	2	1024	3	2.4	512	5	2.4	1024	3	2.4	512	5	2.4	1024	3	2.4	512	5	2.4	1024	3	2.4	512	5	2.4	1024	3	2.4	512	5	0.00	0.00	0.00	0.00
H-H	2.4	1024	4	2.4	1024	4	2.4	512	3	2.4	512	5	2.4	256	3	2	512	4	2.4	512	3	2.4	512	5	2.4	512	3	2.4	512	5	16.04	28.21	16.04	16.00
H-M	2.4	1024	1	2.4	512	7	2.4	256	1	2.4	256	7	2.4	512	4	2.4	512	4	2.4	512	1	2.4	256	7	2.4	1024	1	2.4	256	7	1.12	35.45	0.00	0.75
M-M	2.4	1024	4	2.4	1024	4	2.4	512	4	2.4	512	4	2.4	512	4	2.4	512	4	2.4	512	4	2.4	512	4	2.4	512	4	2.4	512	4	7.22	7.22	7.22	7.22

accepts the features of the co-located (unknown) applications as input. The output of the model is the optimal configuration parameters for the co-located applications to achieve the maximum energy efficiency. Table 1 presents the absolute percentage error (APE) of the studied learning models where all combinations of tuning parameters (number of mappers, frequency and HDFS block size) are explored for co-located applications. LR shows to have the highest APE of 55%. As the model gets more complex, non-linear decision tree model (REPTree) reduces the error significantly. The MLP model has the lowest average APE (0.772 %). We have integrated these machine-learning models into self-prediction techniques, presented in Figure 7.

6.4 Methodology

Our proposed self-tuning prediction (STP) techniques presented in Figure 6 and Figure 7 include Lookup Table-based self-prediction technique (LkT-STP) and Machine Learning Model-based self-prediction technique (MLM-STP).

- LkT-STP Technique

The first step involves constructing a feature matrix of the studied training applications (as explained in the Section 6.1) by evaluating their micro-architectures and real time system resource utilization "i.e." CPU utilization, memory footprint, number of I/O requests (write/read), and IPC. This is done in Step 0 as shown in the Figure 6. Resource profiling identifies the runtime characteristics and resource utilization of MapReduce applications. We have reduced the features metrics to the most vital and distinct micro-architecture parameters via PCA analysis to capture the characteristics of MapReduce application. Additionally, the database contains the optimal configuration parameters providing the minimum EDP (as explained in Section 6.2) for all applications in the training set. Based on the values from the features matrix, we use STP to predict the optimal tuning parameters for unknown incoming MapReduce applications as follows: Let's assume that the testing applications, App_x and App_y, with a specific input data size, are to be run concurrently. In the first step, the feature vectors of a subset of the application are created. This is done by running the application for a learning period and collecting the features information (i.e., LLC MPKI, Branch misprediction, CPU utilization, memory footprint, etc.). Second, the cluster algorithm classifies the testing application based on the feature matrix information. In other words, the classifier chooses the application in the database that best resemble the testing applications. Third, we scan the database to extract the tuning parameters that provide the minimum EDP for the co-located applications.

- MLM-STP Technique

Similar to LkT-STP, in MLM-STP (presented in Figure 7) we construct the feature matrix and store them in a database. Later, we build a machine learning model (LR, REPTree, and MLP) based on the feature matrix for each specific class (C-bound, I-bound, M-bound, H -bound), shown in Step 0 (B). For incoming unknown applications, we first classify them and then we select the model that best suits the characteristics of the applications (Step 3). In Step 4, we run the selected model with all permutations of tunable parameters. Finally, the configuration parameters correspond to the minimum calculated EDP out of the prediction model are selected for application run.

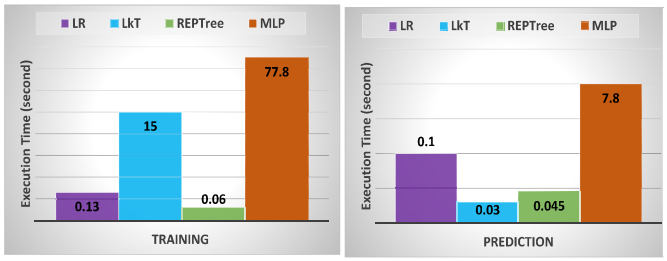

Figure 8: (a) Training (b) Prediction Computational Complexity of the studied algorithms

7. Comparison of Self-Tuning Prediction Techniques

In this section, we validate the proposed STP techniques to find out how effectively they predict the optimal configuration parameters of co-located applications for maximizing the energy efficiency.

For a pair of located applications, given that we study 11 Hadoop MapReduce applications each with 3 different input data sizes, there are 528 different workloads (pair of applications) that can be selected. For each application in a workload, given that the tuning parameters include 5 different HDFS block sizes, 8 different number of mappers, and 4 different operating frequency levels, there are 160 possible cases that need to be examined to find the optimal set of parameters. Therefore, 84,480 application runs (and their associated data sets) are examined in this work to find the best offline tuning parameters for each possible pair. To validate STP, we divide the dataset into two non-overlapping sets; a training set and a testing set.

The training set (based on the known applications) is used to build the database discussed in the Section 6.2. The testing set (based on the unknown applications) is used to validate how well our proposed STP techniques find the optimal tuning parameters for unknown incoming workloads. Note that Naïve Bayes (NB), Collaborative Recommendation Filtering (CF), SVM, PageRank (PR), Hidden Markov Model (HMM), and K-mean (KM) applications are assumed unknown applications and were not used to generate the training dataset. Workload can be a combination of known applications and unknown applications.

Table 3: 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]	[svm, svm, wc,wc, svm, wc, hmm, wc, hmm, hmm, wc,wc, hmm, wc, svm, wc]
WS2	[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]	[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, hmm, svm, ts, st, wc, wc, ts, st]
WS5	[C,H,I,H,C,H,I,H,C,H,I,H,C,H,I,H]	[hmm, ts, st, ts, wc, ts, st, ts, svm, ts, st, ts, hmm, 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, hmm, svm, ts, st, wc, wc, ts, st]

7.1 EDP Error Estimation for Unknown Workloads

We compared proposed STP techniques with the oracle COLAO technique, by calculating the relative EDP difference of the two (we will refer to this as error rate). On average, the error rate for LkT, LR, REPTree and MLP against COLAO is 8.09%, 20.37%, 3.84%, and 3.43%, respectively. LR shows substantially higher error rate compare to others therefore LR is not considered as a good model to configure the optimal parameters. On the other hand, REPTree and MLP error rate is lower than (on average) 4%.

Table 2 presents the error rate and the optimal configuration parameters identified using COLAO and STP techniques (LkT, LR, MLP and REPTree) for a subset of the studied testing workloads. Although the tuning parameters obtained with the COLAO provide the maximum energy efficiency, an extensive set of experiments are required to uncover them. Our proposed techniques can predict the optimal tuning parameters for the unknown incoming MapReduce applications within less than (on average) 4% of the upper bound using REPTree and MLP, respectively. In the worst case, a 16% error with REPTree and MLP machine learning models is still a small price to avoid an exhaustive brute-force search. In other cases, the STP technique achieves energy efficiency close to the COLAO oracle. We achieve this high accuracy despite decoupling decision to co-locate and tuning applications, as proposed in Figure 4.

Overall, the results are quite promising. By decoupling the co-locating and tuning decisions, we significantly simplify the complex optimization problem of searching for the best combination of applications to co-located onto a single node, as well as tedious navigation to find the right tuning parameters for each application. While the decoupling eliminates several possible scheduling and tuning optimization decisions, the approximately 3.84% difference from an offline oracle scheduler shows its effectiveness, while simultaneously making the decision process much simpler.

7.2 STP overhead

In this section we discuss the training time and prediction time of various models proposed to implement STP. It is important to note that while training time is done offline and only once, the prediction time is done at run-time and for every new incoming application, and therefore is considered as a performance overhead for STP.

Figure 8 shows the average training time as well as the prediction time of each STP model. As the complexity of machine learning model increases its training time increases as well. The training time of LR and REPTree (0.13 sec and 0.06 sec) is orders of magnitute lower than the LkT and MLP techniques (15 sec and 77.8 sec). However, as discussed the one-time training process is not a performance bottleneck. A similar trend is observed for the prediction time with the exception of LkT technique. The main reason is LkT is a simple look up table-based model, which selects predetermined configuration parameters from a small table and is the least computationally complex model. Although, MLP error rate is the lowest, its relatively long prediction time makes it a less favourable choice. REPTree and LkT not only have low error rates, their computation complexity are also low. REPTree is the more preferable prediction model compared to LkT as the extensive search is required to populate lookup table with the optimal configuration parameters. In addition, as compared to REPTree model, LkT does not provide any flexibility in selecting the configuration parameters for unknown incoming application. Overall, REPTree is shown to offer the best trade-offs between accuracy, complexity as well as prediction time compared to other predictive models.

8. Scalability

In this section, we evaluate the scalability of ECoST on a local cluster with 1, 2, 4, and 8 node Intel Atom servers. [Not Pair-NP, Not Tune-NT] indicates that we are running applications without tuning their configuration parameters. No pairing is done as applications are running serially.

Application Mapping Policies

We have evaluated the workloads as shown in Table 3, where each workload comprises of 16 applications that can be co-located via a decision tree (Figure 4) and tuned using STP to achieve maximum EDP. With respect to the number of nodes considered 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. Serial Mapping is referred as SM.
2. Multi-Node Level Mapping [NT]: Nodes are divided among running applications. For instance, if we have 8 nodes then we can run 2 applications in parallel each on 4 nodes (MultiNodeLevel1 referred as MNM1) or 4 applications each on 2 nodes (MultiNodeLevel2 referred as MNM2).
3. Single Node Mapping [NT]: Each application is being assigned to a single node (all 8 cores of a node are running the application). Single Node Mapping is referred as SNM.
4. Core Balance Mapping [NT]: Two applications are being co-located on a single node and half the cores (4 cores) are assigned to each application to run. Core Balance Mapping is referred as CBM.
5. Predict Tuning Mapping [NP, T]: In this policy, we do not pair applications. However, we apply STP technique to predict the best configuration parameters to execute each application. Predict tuning mapping is referred as PTM.
6. ECoST [P, T]: This mapping policy presents our proposed technique, ECoST. Similar to Core Balance, two applications are running in parallel on a single node. However, the number of cores assigned to each application is predicted using STP. Furthermore, we have paired the application with respect to the decision tree discussed in Figure 4.
7. UB: Upper bound presents the best pairing and tuning configuration parameters obtained through brute force search for maximum energy-efficiency.

Figure 9 presents the EDP results for randomly selected workload policies with 1, 2, 4 and 8 nodes at the local cluster. All results are normalized to the result of Oracle mapping policy. Serial mapping with no tuning (NT) performs poorly. However, mapping policies (Multi-Node Level1, Multi-Node Level2 and Single Node) that allow multiple applications to run in parallel improve EDP. Furthermore, we have studied the impact of co-located applications at the node level – Core Balance Mapping and ECoST. 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, in the 8 Nodes case, Predict Tuning

Mapping has on average 53% and 55% better energy efficiency as compared to Single Node Mapping and Core Balance Mapping, respectively. In addition, we have observed that our proposed technique ECoST not only performs better than other studied mapping policies, it achieves EDP improvement very close to an upper bound brute force technique. In an 8-node server, our proposed ECoST technique achieves energy efficiency on average within 8% of the upper bound.

ECoST not only allows more MapReduce applications to execute concurrently at the datacenters level by fine-tuning the configuration parameters, but also it achieves close to the best possible EDP found using an offline policy and exhaustive search. We see this across multiple sizes of machines (1, 2, 4 and 8 nodes), despite the increasing complexity of the co-locating applications and parameter configuration space.

9. Related Work

There has been a significant amount of work to address the challenge of co-locating applications on multicore processor [21]. 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-scheduling them along the base vectors for selecting the optimal co-schedules. Many co-scheduling studies on CMP platforms [8, 9] 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. In [26], authors have used L2 cache miss rate predictions to schedule suitable threads together on a CMP platform. In [27], authors model resource interference of server consolidation workloads by estimating cache usage while co-scheduling two jobs at a time. The work in [15] has studied the co-scheduled HPC applications by evaluating the affinity-aware contention information with the greedy allocation heuristics technique. Our work is orthogonal to these resource-awareness techniques. [21] proposes the energy-aware thread-to-core scheduling policy for heterogeneous multicore processor. Our work targets microserver and highlights the fact that Hadoop-based big data applications can also be co-scheduled onto one node by concurrent fine-tuning of frequency and HDFS block size and still remain as energy-efficient as using maximum number of cores.

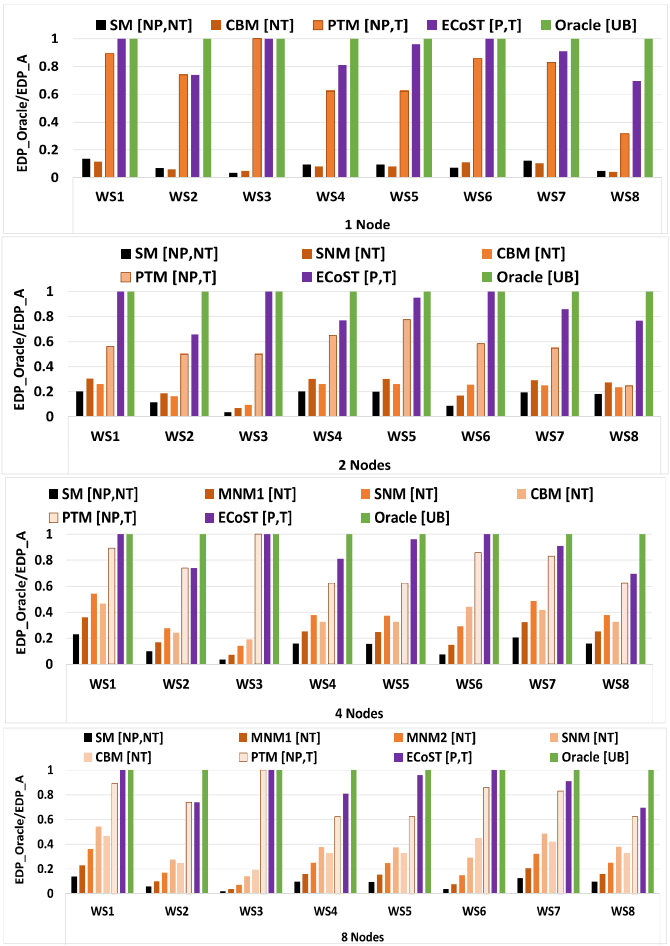

Figure 9: EDP improvement with respect to various mapping policies at (a) 1 Nodes (b)2 Nodes (c)4 Nodes (d)8 Nodes (top to bottom) (A = SM, MNM1, MNM2, SNM, CBM, ECoST)

Big data frameworks and in particular Hadoop-based applications [39] inherent different micro-architectural behavior than traditional application (SPEC and PARSEC) [1, 23]. In addition, these frameworks have large set of tuning knobs, which individually and concurrently influence the scheduling decision. None of above techniques therefore are directly applicable for co-scheduling 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, which cannot capture the real-system behavior of complex big data framework.

Several efforts have sought to reduce the energy consumption of Hadoop clusters, e.g. [13]. They have proposed Covering Set strategy that reduces the energy consumption of clusters by altering the data placement policy in HDFS. The main issue there is that roughly 25% servers has to be in the covering set state and they cannot be powered off, even if they are not utilized for running computations. In addition, this strategy needs understanding on how to place the data replicas in HDFS so that servers can be turned off without affecting the data availability.

Scheduling techniques using linear regression to predict the performance or energy estimation using traditional applications have been addressed in numerous studies [25]. [25] has proposed a simulation-based prediction framework for Hadoop to derive efficient task scheduling using linear regression. Linear regression model, implements a statistical model that assumes a linear relationship between input variables and output variable to obtain optimal results. Without considering the significance of input variable, linear regression model can eliminate any input variable that illustrates non-linear relationship against the output variable. As we showed in this work, linear regression, based model performs poorly in capturing application behavior and finding the best parameters to lower the EDP in co-scheduled MapReduce applications.

Some recent researches have investigated the auto-tuning of MapReduce configuration parameters using machine learning techniques There are others work that looked into other aspects of machine learning to solve other issues in computer design[33-38]. [16] used machine learning techniques for MapReduce workloads to predict the performance by capturing the effects of tuning parameters (number of mappers, amount of RAM etc.) on job execution time. However, this paper does not discuss the impact of parameters on power as well as co-scheduling challenge of Hadoop

MapReduce applications. [14] builds a machine learning based performance model to use it as an auto-tuner for Hadoop applications. Considering that this work has performed their analysis on only two small Hadoop kernels, we cannot generalize their performance model to real-world Hadoop-based applications. Unlike ECoST that study the energy efficiency model for the co-scheduled applications, [14] focuses on the performance model for standalone Hadoop application. [16] and [28] use online classification to estimate interference between co-located workloads that are unlikely to cause interference, however [16] does not auto-tune the configuration parameters. The work in [28] adopts a similar approach to ours; however, it only focuses on performance metric. Different from these two works, our work predicts energy-efficiency. In addition, unlike [28] and [14], our results illustrates that HDFS block size has significant impact on the performance and energy efficiency.

10. Conclusion

Co-locating and self-tuning incoming applications to the datacenter at the node level is a challenging problem, in particular for data intensive applications with their complex and deep software stacks, and many tuning parameters. For instance, for MapReduce applications these decisions are influenced by many tuning parameters at the application, system, and architecture levels such as number of mappers, HDFS block size, and frequency of the core. The large number of tuning parameters in MapReduce provides more opportunity for optimization, but it makes it a challenging problem. When considering separately tuned and configured multiple co-located applications, the search space and complexity of the problem explodes.

This paper presents ECoST, an energy-efficient co-located and self-tuning algorithm for data intensive applications. The ECoST methodology was successfully applied to MapReduce framework. ECoST first examines the impact of tuning parameters and the interplay among them on EDP. 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-locate multiple applications at the node level. Based on the characterization results, we develop a self-tuning prediction technique to determine the optimal tuning parameters at run-time for co-located MapReduce applications. ECoST, by decoupling the decision to co-locate applications and tuning parameters concurrently, significantly simplifies the complex optimization problem of searching for the best combination of applications to co-locate onto a single node, and simultaneously navigating the wide array of tuning parameters for each application and their interplay among them. We studied various machine learning based models to implement energy-efficiency prediction in ECoST. Overall, our experimental results show that while a neural network based prediction method offer highest accuracy, among studied machine learning based models, a decision tree based model offers the best trade-offs between accuracy, power and complexity overhead. In addition, decision tree based model predicts optimal parameters for unknown incoming MapReduce applications fairly accurately and shows energy efficiency within 4% of the upper bound results when co-locating multiple applications at a node level and within 8% of upper bound on an 8-node server

References

1. M. Malik, et al. "System and architecture level characterization of big data applications on big and little core server architectures," in TOMPECS 2018
2. M. Ferdman, et al. "Clearing the clouds: a study of emerging scale-out workloads on modern hardware." In ACM SIGPLAN 2012
3. S. Huang, et al. “The hibench benchmark suite: Characterization of the mapreduce-based data analysis,” in ICDEW 2010
4. Weka: https://www.ibm.com/developerworks/library/os-weka1/
5. Wattsuppro,” https://www.wattsupmeters.com/secure/index.php.
6. Dsat,” http://lintut.com/dstat-linux-monitoring-tools/.
7. D. Doucette, et al. “Base vectors: A potential technique for microarchitectural classification of applications,” in WIOSCA 2007
8. Y. Jiang, et al. “Analysis and approximation of optimal co-scheduling on chip multiprocessors,” in PACT 2008
9. Y. Xie, et al. “Dynamic classification of program memory behaviors in cmps,” in the 2nd Workshop on CMP-MSI 2008.
10. Caulfield, A. M., et al., “A cloud-scale acceleration architecture.”, In MICRO 2016
11. http://www.storageswitzerland.com/Articles/Entries/2013/5/20_What_Is_A_Hyperscale_Data_Center.html
12. Jaleel, et al. “Cruise: cache replacement and utility-aware scheduling,” in ACM SIGARCH Computer Architecture News, 2012
13. J. Leverich et al. “On the Energy (In) efficiency of Hadoop Clusters.” In HotPower, 2009.
14. N. Yigitbasi, et al. "Towards machine learning-based auto-tuning of mapreduce." In MASCOTS 2013
15. S. Kim, et al. “Platform and co-runner affinities for many-task applications in distributed computing platforms,” in CCGrid 2015
16. C. Delimitrou, et al. "Paragon: QoS-aware scheduling for heterogeneous datacenters." In ACM SIGPLAN 2013.
17. C. Wu, et al. "A green energy-efficient scheduling algorithm using the DVFS technique for cloud datacenters." In FGCS 2014
18. Dimitrov, Martin, et al. "Memory system characterization of big data workloads." In Big Data 2013
19. Apache Mahout: scalable machine-learning and data-mining library
20. J. Mars, et al., “Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations,” In MICRO, 2011
21. R. Nishtala, et al., “Energy-aware thread co-location in heterogeneous multicore processors” In EMSOFT, 2013.
22. M. Malik, et al. "Characterizing Hadoop applications on microservers for performance and energy efficiency optimizations," in ISPASS 2016
23. K.R. Krish, et al. “[phi] Sched: A Heterogeneity-Aware Hadoop Workflow Scheduler.” In MASCOTS 2014
24. G. Sabin, et al. "Scheduling of parallel jobs in a heterogeneous multi-site environment." In JSSPP 2003
25. K. Choi, et al "Dynamic voltage and frequency scaling based on workload decomposition." In ISLPED 2004
26. Fedorova, et al. "Performance of multithreaded chip multiprocessors and implications for operating system design." 2005
27. L. Tang, et al. "The impact of memory subsystem resource sharing on datacenter applications." In ISCA 2011
28. C. Delimitrou, et al. "Quasar: resource-efficient and QoS-aware cluster management." In ACM SIGPLAN 2014.
29. S. Che, et al. “A characterization of the Rodinia benchmark suite with comparison to contemporary CMP workloads.” In IISWC 2010
30. R. Appuswamy, Raja, et al. "Scale-up vs scale-out for hadoop: Time to rethink?." In SoCC 2013
31. Li, Sheng, et al. "McPAT: an integrated power, area, and timing modeling framework for multicore and many core architectures." In. MICRO, 2009
32. M. Malik, et al., “Co-Locating and concurrent fine-tuning MapReduce applications on microservers for energy efficiency”, in IISWC’17
33. H Sayadi et al, “Power Conversion Efficiency-Aware Mapping of Multithreaded Applications on Heterogeneous Architectures: A Comprehensive Parameter Tuning” in ASPDAC 2018
34. H Makrani et al XPPE: Cross-Platform Performance Estimation of Hardware Accelerators Using Machine Learning” in ASPDAC 2018
35. H Sayadi et al, Specialized Hardware-Supported Malware Detection Using Machine Learning Techniques in TDSC 2018
36. H. M. Makrani et al. “Energy-aware and Machine Learning based Resource Provisioning of In-Memory Analytics on Cloud”. In SoCC 2018.
37. Hossein Sayadi et al. 2017. Machine learning-based approaches for energy efficiency prediction and scheduling in composite cores architectures. In ICCD.
38. Hossein Sayadi et al. 2018. Customized machine learning-based hardware-assisted malware detection in embedded devices. In TrustCom/BigDataSE. [22]
39. Li, et al. "CloudCmp: comparing public cloud providers." In ACM SIGCOMM 2010.
40. G. Sabin, et al. "Job fairness in non-preemptive job scheduling." In ICPP 2004
41. M. Malik, et al. “Big vs Little Core for Energy-Efficient Hadoop Computing”. in JPDC 2018
42. R. L. Villars, et al., "Big data: What it is and why you should care." in IDC 14 (2011): 1-14.