Creating Soft Heterogeneity in Clusters Through Firmware Re-configuration

Xin Zhan
School of Engineering
Brown University
Providence RI, USA
xin_zhan@brown.edu

Mohammed Shoaib
Microsoft Research
One Microsoft Way
Redmond WA, USA
mohammed.shoaib@microsoft.com

Sherief Reda
School of Engineering
Brown University
Providence RI, USA
sherief_reda@brown.edu

***Abstract*—Customizing server hardware to adapt to its workload has the potential to improve both runtime and energy efficiency. In a cluster that caters to diverse workloads, employing servers with customized hardware components leads to heterogeneity, which is not scalable. In this paper, we seek to create soft heterogeneity from existing servers with homogenous hardware components through customizing the firmware configuration. We demonstrate that firmware configurations have a large impact on runtime, power, and energy efficiency of workloads. Since finding the firmware configuration that minimizes runtime and/or energy efficiency grows exponentially as a function of the number of firmware settings, we propose a methodology called *FXplore* that helps complete the exploration with a quadratic time complexity. Furthermore, *FXplore* enables system administrators to manage the degree of the heterogeneity by deriving firmware configurations for sub-clusters that can cater to multiple workloads with similar characteristics. Thus, during online operation, incoming workloads to the cluster can be mapped to appropriate sub-clusters with pre-configured firmware settings. *FXplore* also finds the best firmware settings in case of co-runners on the same server. We validate our methodology on a fully-instrumented cluster under a large range of parallel workloads that are representative of both high-performance compute clusters and datacenters. Compared to enabling all firmware options, our method improves average runtime and energy consumption by 11% and 15%, respectively.**

I. Introduction

Servers are the workhorse of high-performance computing (HPC) clusters and datacenters. Commodity servers are designed to cater to a large range of applications. As a result, their hardware configurations tend to be geared towards the typical workload. On the flipside, workloads exhibit large variation in their characteristics that could be exploited by designing servers with customized hardware configurations to deliver improved performance and energy efficiency. For instance, while employing a faster DRAM in a server helps reduce runtime of applications that are memory-bound, a slower CPU can help reduce power consumption of those that are network-bound [1], [2]. However, since servers are meant to support diverse workloads in a cluster, making application-specific hardware component changes is impractical. Furthermore, creating a cluster by purchasing servers with different hardware capabilities can complicate resource management and increase costs. Instead, a more realistic approach is to change the configuration of the available hardware and software components.

Among the many potential ways of changing the configuration of the hardware components [*e.g.*, through virtual machine managers (VMMs) and operating system (OS) parameters], we observe that modern servers offer a large number of firmware configurations (*e.g.*, through the BIOS or UEFI) that can be tuned with significant impact on the runtime and power consumption of a server. In fact, some settings like those related to memory and storage can only be tuned through the firmware configuration.

Thus, by configuring firmware differently for different servers, we can create a *soft heterogeneous* mix of servers out of an originally homogeneous set that delivers improved performance and energy efficiency for targeted workloads. In comparison to purchasing custom servers, a soft approach to creating heterogeneity allows us to change customizations with a simple reboot of the servers.

The traditional approach of configuring the firmware involves a human in the loop. System administrators follow simple ad-hoc rules to identify the appropriate firmware settings [3], [4], which can potentially lead to ineffective use of the hardware components and is naturally prone to human errors. In contrast, we propose an automated firmware option exploration tool called *FXplore* that is far more effective in finding firmware configurations of servers that can deliver the maximum benefits in performance and energy efficiency. Figure 1 contrasts the traditional approach with our approach.

There are several challenges in finding the optimal configurations. First, there are an exponential number of configurations as a function of the number of firmware settings, which makes identifying the optimal configuration for a workload a hard problem. Second, creating a dedicated sub-cluster with its own custom firmware configuration for each target workload can complicate system management, especially if there are a large number of target workloads. Third, administrators sometimes deploy co-runners on the same server. Through *FXplore*, we provide a framework that addresses these challenges. We make the following key contributions.

- We quantify the impact of firmware configurations on the runtime and power consumption of a diverse range

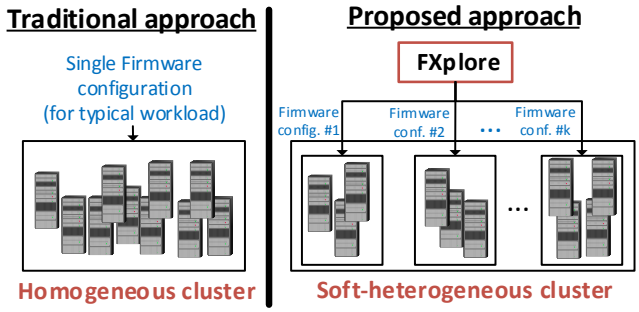

Fig. 1. *FXplore* enables soft heterogeneity in a cluster by customizing firmware for target workloads to improve performance and energy efficiency.

of workloads. We demonstrate that the optimal configurations for these workloads can be very different. Furthermore, we show that the optimal configurations cannot be derived by analyzing the impact of each of the individual firmware settings in isolation.

- We propose an automated firmware configuration exploration methodology called *FXplore* that employs a sequential-search heuristic algorithm to identify the optimal configuration for any given workload with substantial speedups compared to brute-force search. In particular, for every n firmware settings, we show that we can reduce the exploration time from $O(c^n)$ to $O(n^2)$.
- To simplify system management, *FXplore* uses machine-learning (ML) techniques to trade-off the degree of heterogeneity in the cluster with the optimality of the performance and energy. In particular, *FXplore* uses k-means to partition a homogeneous cluster of servers into sub-clusters, each with its unique firmware configuration and suitability for sets of targeted workloads rather than a single workload. Thus, *FXplore* simplifies system management in the presence of heterogeneity.
- We extend *FXplore* to handle co-running workloads, such that it identifies the firmware configurations for cases when multiple workloads are run on the same server. We also evaluate a number of techniques from the literature geared for on-line operation to enable administrators to map new workloads to existing sub-clusters depending on the similarity to workloads in the training set.
- We validate our methodology on a fully-instrumented cluster with eight server nodes using a diverse set of parallel workloads that span HPC applications and datacenter workloads. We demonstrate that *FXplore* can improve runtimes by 11% and energy efficiencies by 15% in average, compared to baseline firmware configurations. It can also accelerate firmware configuration exploration by 2.2× compared to brute-force exploration.

The rest of the paper is organized as follows. In Section II, we motivate the need for *FXplore*. In Section III, we describe our firmware configuration methodology including the sequential exploration, sub-clustering and ML-based mapping approaches. In Section IV, we present the experimental results. In Section V, we discuss related work in the context of our results, and finally, we conclude in Section VI.

II. Motivation: Impact of Firmware Configuration

In this study, we aim to improve the performance and energy efficiency of a server by customizing its hardware to the software characteristics of the workloads using the options in the firmware. Modern servers offer many firmware configurations with each offering a different impact on the power-performance levels of different workloads.

Table I lists five important firmware settings that are available in our servers through the BIOS. There are two settings related to the cache performance: the hardware prefetcher (HP) and adjacent cache-line prefetcher (CP). HP enables prefetching between the cache and main memory, while CP enables prefetching between cache and CPU cores. Enabling HP and CP will benefit workloads with predictable memory-access patterns and good data locality. CPU turbo boost (CT) mode is another option that is also widely available in server-class processors. CT enables processors to operate at a higher frequency than the nominal especially when thermal and voltage slacks are available. The speed of memory is another configuration option that can be tuned from firmware. Our servers provide control over the frequency through an option called the memory turbo boost (MT). Enabling MT allows the memory to run at a higher frequency (1066 MHz), while disabling it lowers the speed (to 800 MHz). Thus, by enabling MT, the system can potentially lower cost for memory accesses, which will provide a sizable benefit to memory-bound applications. Hyper-threading (HT) allows throttling the server performance through simultaneous multi-threading. By sharing the hardware resources on a single physical processor core, two different threads can be executed simultaneously. Ideally, multi-threaded workloads may benefit from enabling HT at the cost of some additional power. However, for workloads with compute-intensive threads, enabling HT can, in fact, create contention for the CPU core and degrade performance compared to no HT. Thus, optimally configuring these five firmware options is a highly nuanced and workload-driven task.

Firmware Setting	Description
Hardware prefetcher (HP)	Enabling HP fetches the data and instructions from memory into cache before the processor loads them.
Adjacent cache-line prefetcher (CP)	Enabling CP will make the processor always fetch two adjacent cache lines.
CPU Turbo boost (CT)	Allows CPU cores to scale up their clock frequency on-demand depending on thermal or voltage slack.
Memory Turbo boost (MT)	Allows adjustment of the memory frequency to a higher or lower value.
Hyper threading (HT)	Enables simultaneous multithreading, which allows threads to share the processor resources where each physical core is regarded as two virtual cores.

TABLE I
Firmware settings that we explore in our study.

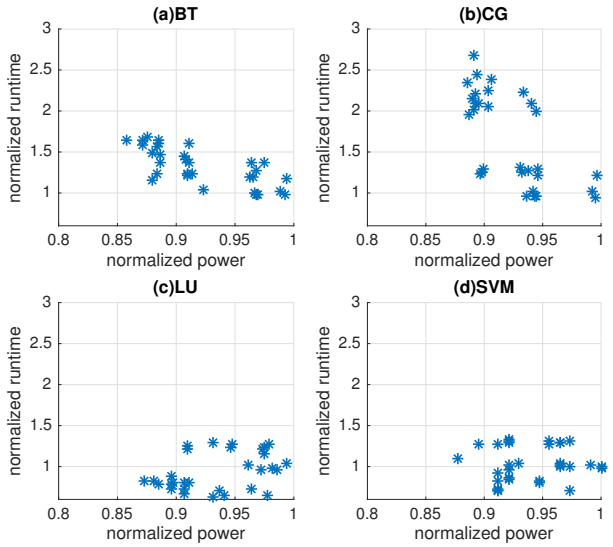

Fig. 2. Firmware configurations can have a large impact on the runtime and power consumption of a server depending on the application characteristics.

Among the limited set of options provided by the version of the firmware on our servers, we found that the above five settings are the ones that impact power and performance levels the most. Thus, we focus on these five parameters to demonstrate the benefits of *FXplore*. However, with newer firmware versions and server models, there are many more configuration options that are available [4]. Such options provide an opportunity for much finer grain firmware tuning by *FXplore*. The basic premise of our work lies in the fact that enabling or disabling firmware options can have a large impact on the performance and power consumption of servers as a function of the workloads. To illustrate this point, we profile a number of parallel workloads on our networked cluster that is composed of eight fully instrumented Xeon-based servers. We select parallel workloads from the NAS Parallel Benchmarks (NPB), which are representative of HPC and workloads from the NU-MineBench benchmarks, which are representative of data mining workloads. Since we consider five firmware options, each server in the cluster can be configured in total of $2^5 = 32$ different ways. Since we run an application on the entire cluster, we enforce any chosen firmware configuration on all of our servers. For instance, consider four exemplary workloads from our two benchmark datasets. Figure 2 shows the runtime and power of these workloads under the 32 possible firmware configurations. All results are normalized with respect to the results when all firmware options are enabled. The trends in the figure lead us to our first observation.

Observation #1:. Firmware settings can have a large impact on the performance and power consumption of applications since each application has its own unique characteristics. For instance, from Figure 2, we see that application `CG` shows 173% variation in runtime as a function of the firmware settings. However, application `SP` shows 59% variation in runtime as a function of the firmware settings. Given these large application-dependent variations in runtime and power,

	Runtime-optimal Configurations						Energy-optimal Configurations					
NAME	CT	MT	HT	HP	CP	Runtime	CT	MT	HT	HP	CP	Energy
BT	✓	✓	✓	✓	×	0.982	×	✓	✓	×	×	0.962
CG	✓	✓	✓	✓	×	0.937	×	✓	✓	×	×	0.915
LU	✓	✓	×	✓	×	0.629	×	✓	×	×	✓	0.569
SVM	✓	✓	×	✓	✓	0.701	×	✓	×	×	✓	0.655

TABLE II
OPTIMAL FIRMWARE SETTINGS ARE DIFFERENT FOR EACH OF THE FOUR APPLICATIONS CONSIDERED IN FIGURE 2. FURTHER, DIFFERENT SETTINGS OPTIMIZE RUNTIME AND ENERGY CONSUMPTION.

it is imperative to ask if there are any shared optimal settings among various applications or whether there is a simple characterization for the optimal runtime and energy efficiency based on the firmware settings.

Observation #2:. The optimal firmware configurations vary by application, where each application could have its own distinct optimal configuration for performance and energy efficiency. Table II illustrates how the optimal configurations for minimizing runtime and energy consumption of the four workloads can be drastically different. The results interestingly show that enabling all firmware settings does not necessarily lead to the best runtime or energy efficiency. One possible reason for this behavior is that the enabled options could conflict with each other. Further, different applications have different sensitivity to the firmware settings. Turning on some firmware settings might not yield good performance improvement or even not have a positive impact. For instance, enabling HT reduces the runtime of many workloads, but it hurts `LU` and `SVM`. The results also show that the optimal settings for runtime minimization may not be the best settings for energy minimization.

Observation #3:. There are subtle interactions among the firmware configurations that do not necessarily add up to provide or diminish gains. Thus, combining settings that yield better results individually may not necessarily further improve the results, and similarly combining settings that have a negative impact individually might surprisingly lead to a positive impact. Figure 3 illustrates the inter-dependency between MT and HP. In the figure, we compare the runtime under three different firmware configurations: enabling only MT, enabling only HP, and enabling both MT and HP. The runtime is normalized against the case that all five settings

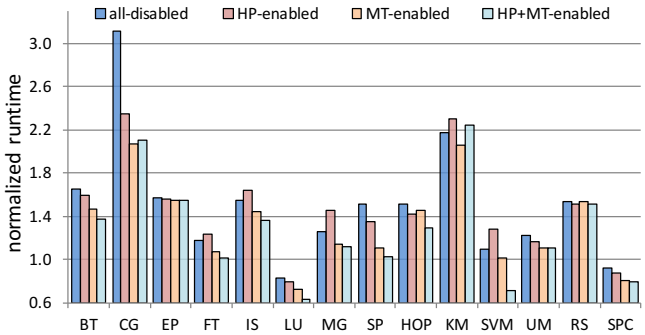

Fig. 3. The normalized runtime of workloads under firmware configurations with only HP enabled, only MT enabled, and both HP and MT enabled shows how the interdependence between firmware options is application specific. The baseline (*i.e.*, runtime = 1) case is where all five options are enabled.

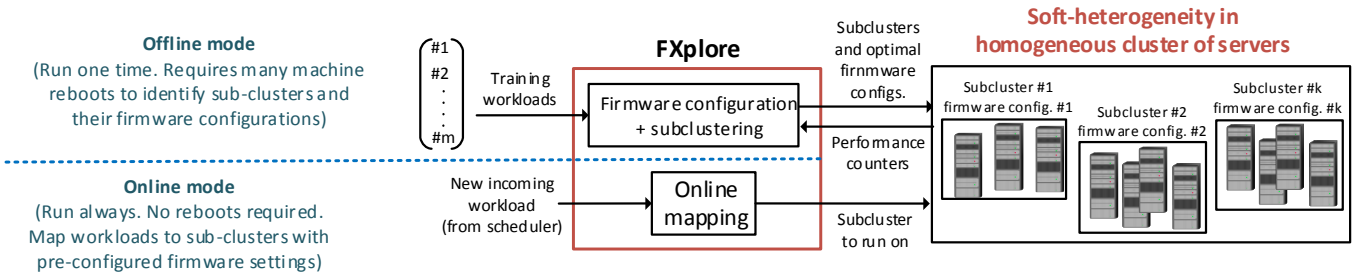

Fig. 4. Overview of *FXplore*. It comprises two operation modes. First, is an offline mode where we use sequential search and sub-clustering to find the optimal firmware configurations for groups of servers. This requires rebooting the servers several times. Second, is the online mapping stage, where we use machine learning algorithms to map incoming workloads to appropriate sub-clusters requiring no rebooting of the servers.

are enabled. The runtime of disabling all firmware options is also plotted for reference to show the impact of the individual options. From the results, for most of the applications such as `BT`, `LU`, `SP`, `HOP`, and `SPC`, enabling HP and MEM individually improves the runtime over all-disabled case and enabling both of them delivers greater improvement. However, for `CG` and `KM`, enabling both of them yields worse runtimes compared to the cases where only MT is enabled. And for `FT`, `IS`, `MG`, and `SVM`, although only enabling HP makes the runtime worse than the all-disabled, when HP is enabled together with MT, the runtime is improved, comparing to the case where either HP or MT are exclusively enabled.

We thus conclude that there is no ideal firmware configuration that is optimal for all kinds of workloads. Also, there is no simple rule that could give us the optimal firmware configuration for a given application. Thus, an intelligent and efficient method to determine the optimal configuration for any given application is highly desirable. Next, we present *FXplore* that precisely addresses this problem.

III. FXplore Methodology

In this section, we present the methodology of *FXplore*. We describe how it enables system administrators to partition servers into sub-clusters such that each sub-cluster is configured with a different firmware configuration to improve performance and/or energy efficiency for target workloads. Figure 4 gives an overview of our methodology. There are two modes of operation. First is the offline mode, where given a set of M target workloads, we partition them into κ sub-clusters, afforded by the administrator, and derive the optimal configurations for the sub-clusters in a fast and effective manner. Second is the online mode, where incoming workloads are profiled using hardware performance counters (PMCs) and mapped to one of the existing sub-clusters with a pre-determined optimal configuration. This process requires no reboots of the servers and is invoked during regular use of the servers. Note that if $M = \kappa$ then each workload can afford its own sub-cluster with optimal settings. However, it is expected that $\kappa < M$ because a large κ can complicate cluster management.

Today, for system administrators, the only available alternative to *FXplore* is brute-force enumeration, where the outcomes (*i.e.*, runtime and energy) under all possible firmware settings need to be enumerated for every workload. Among these settings, the one that gives the best results needs to be chosen and used for the server. For N firmware options, brute-force enumeration requires an exploration of 2^N settings per workload (*e.g.*, $32 \times M$ reboots of the server are required for M workloads and 5 firmware options), which is not scalable especially when modern high-end servers provide 10-15 firmware options that could impact the power/performance of applications [5]. Through *FXplore*, we propose to reduce this firmware exploration time complexity from 2^N to $O(N^2)$. Further, once the one-time exploration process is completed offline, we provide a methodology to map new incoming workloads in real time to sub-clusters of servers with pre-determined optimal configurations. We describe the details of our offline heuristic exploration and sub-clustering methodologies in Sections III-A and III-B, respectively. The details of the on-line operation are given in Section III-C. In Section III-D, we extend our methodology to handle the case when co-runners are enabled; *i.e.*, when multiple applications are run on the same server.

A. Identifying the Optimal Firmware Configurations

This is the first part of the offline process in *FXplore*, which needs to be run one time with $O(N^2)$ server reboots. In this method, we follow an iterative sequential approach to optimize our cost function (*i.e.*, performance or energy) with the aim to minimize the search space of firmware options. Our approach seeks to capture the subtle interactions of the firmware configurations as outlined in observation #3 in Section II, which concluded that the improvements in performance or energy obtained by a combination of firmware options is not equal to a simple superposition of improvements due to each option. The procedure for our proposed sequential search method, *FXplore-S*, is given in Algorithm 1. At the outset, we enable all candidate firmware options and label them as *free* (step 1). Then, in each iteration (step 2), we profile the input workload by temporarily disabling one *free* firmware option at a time (steps 3, 4, and 6). During this time, for each option, we also measure and register the cost function *i.e.*, runtime or energy consumption (step 5). After this, we disable the option that, when it is disabled, the cost function is minimized. We also label that option as *locked* for all subsequent iterations of the procedure (item 7). At the k^{th} iteration, $N-k+1$ *free* options are evaluated and the one with the highest impact on the cost function is disabled and locked. Thus, to disable and lock N firmware options, we will need N iterations. After completing N iterations, we rank the results from all

Algorithm 1 FXplore-S: Offline sequential-search algorithm to determine the optimal firmware configuration.

Input: Workload and N candidate firmware options
Output: Optimal configuration for input workload

1: **initialize** Enable all N options and label them as *free*
2: **for** k =1 **to** N **do**
3: **for each** *free* firmware option **do** // *Option locking*
4: Disable the firmware option
5: Run workload, measure, and record runtime/energy
6: Enable the firmware option
7: **end for**
8: Disable option that, when disabled, allowed the workload to achieve best results in Step 5 and label it as *locked*.
9: **end for**
10: Repeat steps 2-8 until all options are disabled.
11: Search the shortlisted optimal configurations and find the one that gives the best result overall.

iterations by their cost-function value, and set the combination of firmware options that globally minimize the cost function.

The exploration time complexity of *FXplore-S* is $N+(N-1)+\cdots+1 = O(N^2)$, which is a substantial improvement compared to the exponential complexity (*i.e.*, 2^N) of brute-force enumeration. Our approach can also be extended to handle non-binary firmware options. We convert a non-binary firmware option to a group of binary options and configure them using *FXplore-S*. However, during the locking stage, we lock the group as a whole instead of considering the binary options in the group separately.

B. Deriving the Sub-clusters

This is the second part of the offline process, which also needs to be run one time during configuration of the servers. If the system administrator can afford a dedicated sub-cluster for every target workload, then the firmware configurations identified by *FXplore-S* can be directly used. In reality the number of sub-clusters is likely to be smaller than the number of target workloads. Thus, in this section we propose a method, *FXplore-SC* that aims to partition the target workloads into groups with consistency so that they can be executed together in the same sub-cluster configured with the same firmware configurations.

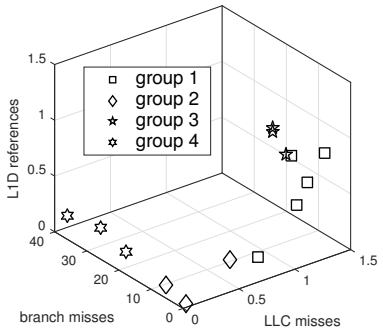

Fig. 5. Plot of truncated feature vectors (only three performance counters out of the five) and their natural clusters.

FXplore-SC proceeds by (1) grouping applications based on their system-level performance characteristics, (2) identifying the optimal configurations for a representative application from each group using *FXplore-S*, and (3) applying the firmware configurations from the representative application of each group to the remaining workloads in the group. The insight behind this procedure is that workloads that exhibit similar system-level performance characteristics should have similar firmware settings.

To measure similarity of workload characteristics, we resort to PMCs. PMC values reveal subtle characteristics of the workloads since they are directly associated with their hardware interactions. For each workload, we compute the average PMC values over time and then per core. For our benchmarks, we collected 13 PMCs covering a diverse range of characteristics. We then computed the principal component analysis (PCA) of the PMC measurements and analyzed the PCA scores of the performance counters. The PCA analysis reveals that the most relevant performance counters are the number of instructions retired, L1 data references, L2 data-cache misses, last-level cache (LLC) misses, and mispredicted branch instructions. As it is difficult to visualize high dimensional data, we selected three of the five counters: LLC misses, branch prediction misses and L1 data cache references and plotted their three-dimensional feature vectors in Figure 5. In the figure, the PMC values are normalized. We can clearly see clustering of the benchmarks even in this lower-dimensional space. *FXplore-SC* is based on PMC values. Its procedure is shown in Algorithm 2. Given a set of workloads, we first quantify the characteristics of each workload in the set by running it on a baseline configuration (*e.g.*, where all firmware options are enabled). We do this quantification by collecting the PMCs mentioned above (step 1). Thus, the average PMC values comprise the feature vector for every workload (step 2). After this, we apply the k-means algorithm to group the workloads into κ groups (step 3). The clustering algorithm works by minimizing the distance between the group members (*i.e.*, workloads) and maximizing the distance between the group centroids. For each group (step 4), we pick a representative workload and

Algorithm 2 FXplore-SC: Offline process to determine sub-clusters of servers and fix their optimal configurations.

Input: Workload set and desired clustering granularity.
Output: Optimal configurations for the subclusters.

1: **initialize** Run every workload on the baseline configuration and collect its PMC values.
2: Average the PMC values for each workload in the set to produce a corresponding unique feature vector.
3: Apply k-means clustering on the feature vectors to partition the workloads into κ groups.
4: **for each** group ε **do**
5: Pick random workload from the group; apply FXplore-S to determine its optimal firmware configuration.
6: Use this optimal firmware configuration for the remaining workloads in the group.
7: **end for**

determine its optimal configuration using *FXplore-S* (step 5). Finally, we employ these settings for all other workloads of the group (step 6).

The value of κ is chosen by the system administrators, which provides them with the flexibility to trade-off the amount of heterogeneity and improvement in performance/power against the amount of management overhead. At its extremes, $\kappa = 1$ leads to one nominal firmware configuration that is used for all servers, and $\kappa = M$ provides a high degree of customization, where every application gets it own sub-cluster that is optimally configured according to its characteristics. It is up to the system administrators to decide the acceptable amount of heterogeneity in the cluster. We will evaluate the impact of κ in Section IV.

C. On-line Operation

This is the online process in *FXplore*, which is run all the time and it requires no server reboots. Once the sub-clusters of servers have been identified and set to have optimal configurations using a number of representative or training workloads, the mapping of a new workload is done based on the similarities to the training workloads. A number of works in the literature tackle this problem [6], [7]. For instance, Liao *et al.* evaluate a number of ML techniques, such as decision trees (DT) and support vector machines (SVM), to map new workloads to a target set of trained workloads as a function of the features of the workloads [6]. We follow a similar approach in this paper. In our case, a new workload is profiled on a baseline server to get its PMC-based feature vector. Then it is mapped using nearest-neighbor (NN) search into one of the existing groups based on the distance between its feature vector and the centroids of the groups, where the cluster with minimum distance is chosen. Once, we map a new workload to a cluster, we use the configuration of the cluster for the incoming workload.

D. Workload Co-location

In many modern clusters, a single server might host multiple workloads concurrently to leverage the availability of the large number of cores or processor sockets. A number of prior works consider the interference arising from allocating workloads on the same server and provide techniques to identify the optimal pairing of co-running workloads to minimize the degradation in performance [7], [8], [9], [10]. We address the co-runner problem by leveraging these prior works to extend our framework so that co-runners are enabled as follows.

1) A desired workload allocation algorithm is first run to identify the optimal pairing of application co-runners on a server with a baseline firmware configuration.
2) If (w_1, w_2) are identified to be an optimal pairing of two applications w_1 and w_2, then *FXplore-S* is executed to identify the best firmware configuration for the pair simultaneously.
3) If clustering is required, then the average per-core PMC vector from running both workloads is first profiled. This average vector is then used as part of the feature space in the sub-clustering algorithm *FXplore-SC*.

We evaluate the effectiveness of *FXplore* in handling co-runners in Section IV-D. Note that we do not propose any new workload co-location or scheduling algorithm. Instead, we observe that co-runner algorithms can have cyclic dependencies with firmware tuning; *i.e.*, there is a possibility that the results of the co-runner algorithm depend on the firmware configuration chosen in the first place. We broke that dependency by choosing to identify the optimal co-runners on a baseline server using existing co-runner algorithms; however, the possibility of co-optimizing the firmware configuration and co-runner pairs of applications can lead to further improvements in future work.

IV. Experimental Results

We present experimental results that validate the ability of *FXplore* to determine the runtime- and energy-optimal firmware configurations for various workloads, while providing system administrators with the ability to control the degree of soft-heterogeneity through sub-clustering. To evaluate *FXplore*, we use a cluster of 8 Dell PowerEdge C1100 servers. Each server is equipped with dual Xeon L5520 quad-core processors (total 8 cores) and 40 GB of DRAM. They run Ubuntu 12.4 and AMI BIOS version 2.66. We use the `perfmon2` tool to collect PMC values from the servers. To determine the total power consumption of the cluster, we created a measurement environment that senses the current flow through the power cord of each server.

We profiled a large set of benchmarks to evaluate the effectiveness of *FXplore*. In particular, we consider workloads from the following two sets of parallel benchmarks: (1) eight workloads from NPB representing computational fluid-dynamics applications: `BT`, `CG`, `EP`, `FT`, `IS`, `LU`, `MG`, and `SP`) [11] and (2) six workloads from NU-MineBench representing data mining applications: `HOP`, `kmeans(KM)`, `SVM`, `Utility Mining(UM)`, `RSearch(RS)` and `SaclParC(SPC))` [12]. NPB is designed to characterize HPC clusters, while NU-MineBench comprises datacenter-like workloads. We configure each workload as 64 threads (8×8) to execute on our experimental cluster. We double the number of threads when hyper-threading is enabled.

To establish ground truth for the optimal configurations, we exhaustively search through all possible firmware configurations. For every configuration, we collect PMC values, power consumption and the runtime of each workload, and compute the energy consumption and exploration time for exhaustive search. Note that to collect the PMC values in our experiments, we run our parallel workloads to completion. We next assess the effectiveness of *FXplore*.

A. Sequential Search Results

In this subsection, we present experimental results for the offline exploration mode of *FXplore i.e.*, *FXplore-S*. We demonstrate the effectiveness of *FXplore-S* in finding the optimal configurations for a given workload, while attaining large reductions in exploration time compared to brute force. We consider the following configurations:

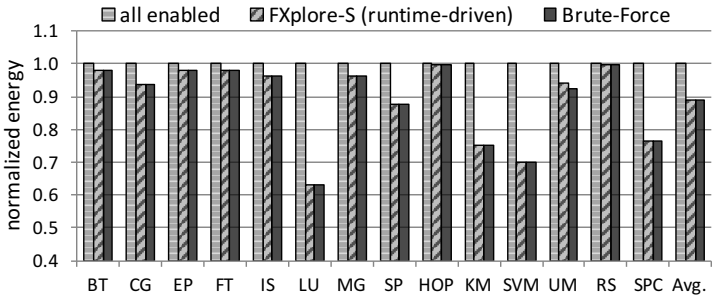

Fig. 6. Normalized runtime improvement of workloads with different firmware configuration methods. Normalized to all-enabled.

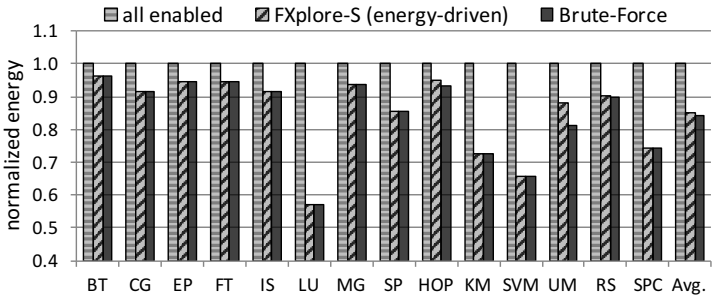

Fig. 8. Normalized energy improvement of workloads with different firmware configuration methods. Normalized to all-enabled.

1) **all-enabled**: We enable all the five firmware options. The runtime (or energy) under other firmware configurations are normalized with respect to this runtime (or energy). We choose it as baseline configuration in all experiments of Section IV.
2) **FXplore-S**: We run our sequential search algorithm to find the best configuration. We consider two cases: a *runtime-driven case* that seeks to minimize runtime, and an *energy-driven case* that seeks to minimize energy.
3) **Brute-force**: We use a brute-force enumeration on all configurations to find the optimal configuration for optimizing runtime or energy consumption.

Optimizing runtime: We report the normalized runtime of all workloads using the different configuration methods above in Figure 6. From the results, we can draw the following conclusions. Enabling all options does not necessarily deliver near-optimal results as is obvious in the cases of `LU`, `KM`, `SVM`, and `SPC`. One possible reason for this behavior is that the enabled firmware options conflict with one another. For instance, if a workload contains several spin-wait loops, enabling HT can create memory-order conflicts and slow down execution. Now, in such a case, if we enable HP in addition to HT, the server can end up with a large number of unusable memory fetches, which may lead to thermal issues and throttling. Thus, the overall performance of the workload may suffer substantially. Figure 6 shows that *FXplore-S* always finds the optimal or near-optimal firmware configuration for all the workloads except for the runtime of `UM`, which is 2% longer than the optimal. *FXplore-S* finds the exact optimal configurations for the rest of the workloads. Overall, *FXplore-S* improves runtime by 11% over the `all-enabled` configuration on average. In terms of exploration time, the results in Figure 7 demonstrate that the average exploration time of *FXplore-S* is only 46% of brute-force search; that is, *FXplore-S* speeds up the exploration time by 2.2×. Note that the speed-up will increase with more firmware settings because the exploration time of *FXplore-S* grows quadratically, while that of brute-force search grows exponentially.

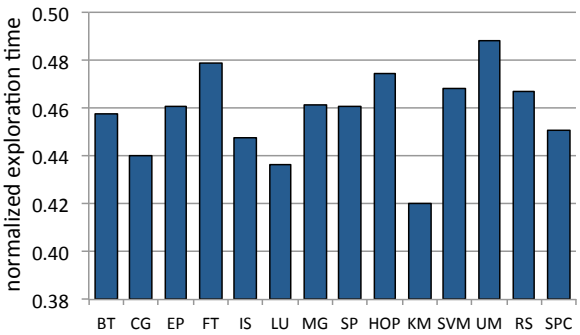

Fig. 7. The normalized exploration time of each workload, normalized to `Brute-force`.

Optimizing energy efficiency: Energy efficiency is another important objective for clusters. Therefore, in this second experiment, we switch *FXplore-S* to the energy-driven case where we determine firmware configurations that minimize the energy consumption for various workloads. For the five options, unlike the impact of the firmware configuration on runtime, the effects on power consumption are more or less predictable. In particular, disabling any of the five options always saves power. Since energy is the product of power and runtime, the impact of the firmware configurations on energy is determined by how pronounced are the savings in power. Figure 8 gives the energy consumption of the workloads for various firmware exploration methods, where we normalize the energy numbers with respect to the energy measurements from `all-enabled`. From the figure, we observe that by using *FXplore-S* we can almost always find the energy-optimal or near energy-optimal configuration. The speed-up in exploration time in the energy-driven case is similar to the runtime case.

Scalability of FXplore with number of firmware settings. Since *FXplore-S* is a heuristic algorithm. In this section, we study its effectiveness and scalability in finding the optimal configuration as a function of the number of available settings N. Since our evaluation servers provide only a limited number of firmware options that we can tune, we consider the cases of $N = 2$-5. We use *exploration error* as a metric to evaluate the effectiveness of *FXplore-S*. For any workload N, let $T_o(N)$ be its runtime under its optimal configuration (found by brute-force search) and $T_s(N)$ be the runtime under the firmware configuration identified by *FXplore-S*. Then, we define *exploration error* as:

$$exploration\ error(N) = \frac{T_s(N) - T_o(n)}{T_o(N)}$$

and *exploration accuracy* as equal to 1 minus the *exploration error*. We report this number using the y-axis on the right hand

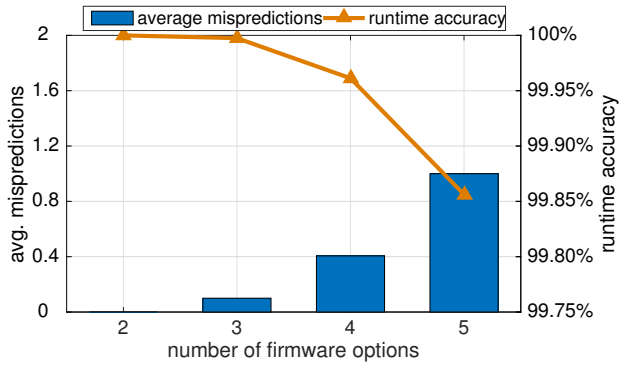

Fig. 9. *FXplore-S* only shows a slight linear increase in estimation error with increasing number of firmware options.

side of Figure 9. We observe a near negligible degradation in exploration accuracy as we go from 1 to 5 firmware options. On the y-axis on the left hand side of Figure 9, we also report the average number of mispredictions that we get for all the candidate workloads using *FXplore-S*. Again a small degradation occurs because of the increase in depth of the search path; however, the inaccuracy is considered almost negligible from a practical perspective. Unlike the cases of $N = 3$ and $N = 4$ firmware options, where we have $\binom{5}{3} = 10$ and $\binom{5}{4} = 5$ estimation samples, respectively, the case of $N = 5$ options has only one sample, which makes it sensitive to experimental noise. The overall trend is that exploration error grows linearly, which means the accuracy will not decrease drastically when more firmware options are considered.

B. Clustering Results

In this subsection, we evaluate the second part of the offline mode in *FXplore*, *i.e.*, *FXplore-SC*. In particular, we assess how well can *FXplore-SC* derive sub-clusters with heterogeneous configurations. The number of sub-clusters κ is a configurable parameter and is chosen by the system administrator depending on management costs. Since we have 14 benchmarks, we can potentially vary κ from 1 to 14 and evaluate the resulting runtimes of the workloads under different clustering results. Note that when $\kappa = 14$, the *FXplore-SC* method becomes essentially the *FXplore-S* method since no clustering is involved, and each workload gets it optimal configuration. However, choosing the optimal number of sub-clusters in general is a challenging task in itself. Choosing a small number of clusters will minimize the heterogeneity of the servers, while a large number of them will complicate system management. Thus, to maintain a good tradeoff between management overheads and the amount of heterogeneity, we choose four sub-clusters for our experiments.

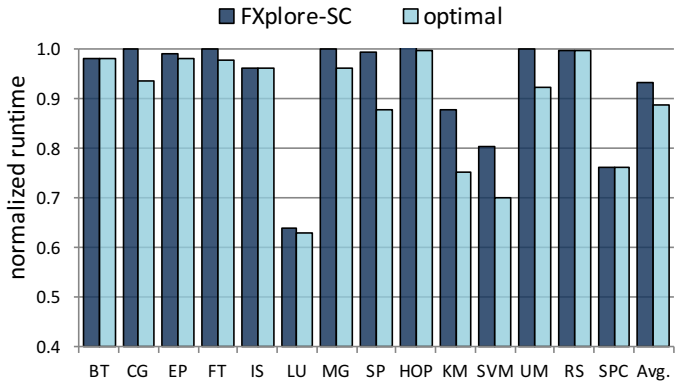

Fig. 10. Workload runtime under sub-clusters firmware configurations created by *FXplore-SC*, when $\kappa = 4$, *i.e.*, four sub-clusters.

name	runtime (×) Classification Method: NN (our approach, FXplore)	SVM	DT	runtime (×) Optimal
BT	0.982	1.371	1.371	0.982
CG	0.937	2.101	2.101	0.937
EP	0.982	0.982	1.547	0.982
FT	1.000	1.000	1.010	1.000
IS	0.962	1.359	1.359	0.962
LU	0.629	0.629	0.629	0.629
MG	0.968	1.116	1.116	0.968
SP	1.029	1.029	0.879	0.879
HOP	1.043	1.043	1.292	0.999
KM	1.161	2.242	2.242	0.908
SVM	0.711	0.711	0.982	0.711
UM	0.999	1.103	1.096	0.999
RS	1.000	1.000	1.514	1.000
SPC	0.789	0.789	1.038	0.789
Average	**0.942**	**1.177**	**1.298**	**0.910**

TABLE III
EFFECTIVENESS OF DIFFERENT MACHINE-LEARNING ALGORITHMS IN MAPPING NEW WORKLOADS TO SUB-CLUSTERS

Figure 10 shows the runtime of each workload in four sub-clusters, *i.e.*, $\kappa = 4$. Again we normalize runtimes to the all-enabled firmware option. In the figure, we compare the normalized runtime of each benchmark under the optimal configuration of the sub-cluster it belongs to and its own individual optimal configuration (determined by brute-force search). We observe that the average runtimes of the workloads under optimal sub-cluster level firmware configurations are only 5% higher than under individually optimal configurations.

C. Evaluation of On-line Mapping Methods

In this subsection, we evaluate the on-line mode of *FXplore*, where incoming workloads from a scheduler get mapped to sub-clusters that are statically pre-programmed with different firmware configurations. For this mode, we leverage some of the existing ML techniques in workload mapping [6], [7], [8]. For high statistical confidence, we employ leave-one-out cross validation of the workloads. Accordingly, we use the offline mode of *FXplore* for all but one workload to determine the sub-clusters and their optimal configurations. Then, we use the PMC-based feature vector of the isolated workload to map it to the sub-clusters using different ML techniques such as DT and SVM, which have been tested in [6] as well as NN, which is used in *FXplore*. We repeat this process to isolate every workload in our test set. Our results presented in Table III show that NN works as well as any other classifier and there is no best mapping algorithm. Overall using NN-based online mapping and four sub-clusters, we find that there is an average of 3% discrepancy in runtime compared to the optimal.

D. Results with Workload Co-location

In this subsection, we show that *FXplore* works for mixture of workloads allocated on the same server. Note that we do not

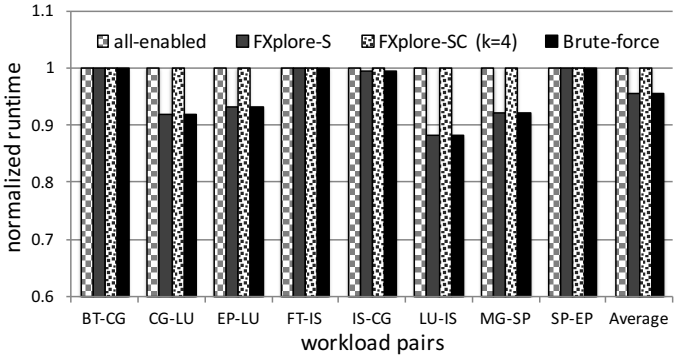

Fig. 11. Avg. runtime of co-located workloads under different firmware configurations, normalized to their average runtime under the baseline configuration.

propose workload co-location or scheduling algorithm. But, rather we study the impact of *FXplore*-based mapping and firmware configuration when we have a mixture of workloads running on a server. To simulate co-located workloads in a realistic setting, we configure each benchmark to occupy four threads per server. Since the first step is to run a co-runner interference algorithm to identify good pairings, we profiled all possible combinations of NPB benchmarks to find, for each benchmark, which other benchmark offers least interference when co-located on the same server. By profiling all possible pairs, we identify the optimal pairs and reduce the "noise" that can be introduced in the experiment if a heuristic interference method is used. We have 8 pairs of benchmarks and run them on all the firmware configurations to evaluate the ability of *FXplore* to deal with a mixture of workloads. Figure 11 gives the average normalized runtime of the two co-located workloads under different firmware configurations, normalized to the runtime of benchmarks co-running under the baseline firmware configuration. The results show that *FXplore-S* can always find the optimal settings. If four subclusters are desired, then *FXplore-SC* results in only a 4.4% increase in runtime compared to the optimal case when each pair gets their own optimal configuration.

Compared to the case of a single workload, we have found that *FXplore* tends to enable more firmware settings when there are co-runners. The reason is that optimal pairing usually pairs workloads that have distinct characteristics to reduce interference; for example, a co-runner pair might include a CPU-intensive application and a memory-intensive application to reduce competition for hardware resources and have minimum interference with each other. However, pairing the workloads in this way will average their performance characteristics and make the combination of them both CPU-intensive and memory-intensive, which leads to enabling more firmware options. However, in many cases, there are possibly subtle interactions between the firmware setting preferences and co-location interference. For example, it might be better to run a workload under its own optimal configuration with a sub-optimal co-runner, instead of scheduling it together with an optimal co-runner on a server with a sub-optimal configuration. A future direction is to find the best matching between servers and workloads after creating the heterogeneity.

V. Related Work

Since applications often have varying hardware-resource requirements, heterogeneity helps improve the performance and energy-efficiency of large-scale computing platforms. In the literature, employing heterogeneous hardware resources has been shown to improve the performance of cloud-computing clusters [7], [8], [9], [13], [14]. Specifically, in [13] and [9], Mars *et al.* have quantified the potential performance improvements that can be achieved by smartly leveraging the heterogeneity in the instruction-set architecture and hardware resources of servers. Subsequent works such as [8] and [7] have proposed methods that employ in-place continuous workload profiling techniques and scheduling to better utilize heterogeneous computing fabrics in warehouse-scale computers. Despite these benefits, cluster operators usually tend to purchase servers with homogeneous hardware configurations in order to minimize cost and resource-management issues. Thus, the existing heterogeneity in server clusters and datacenters is minimal. Consequently, the attainable performance benefits using the aforementioned approaches are limited. Our work is complementary to these existing techniques since it allows us to increase the level of soft-heterogeneity in servers by simply altering their configurations.

Our work focuses on changing hardware-resource configurations of a server through the firmware. Many hardware configurations (*e.g.*, memory and storage settings) cannot be changed *via* the OS or VMMs. There exists a large amount of prior work that employs software techniques such as voltage and frequency scaling of the processor [2], [15], [16] and main memory [17], [18], [19]. Although these are effective methods to control the server behavior, we believe there are many more control knobs to improve or impair server performance and energy efficiency, which can only be controlled through the firmware. CPU hyper-threading is one such example, which is shown to significantly impact application performance [20], [21]. It is important to note that our methodology applies to configure even those options (*e.g.*, CPU and cache settings) that can typically be changed in software.

When it comes to related workload mapping techniques, there is some work in the compiler community [6], [22] that employs ML to tune memory prefetch settings in software and in firmware. [23] provide an overview of some of these approaches. Particularly interesting is [6], where Liao *et al.* adjust four memory prefetch settings and collect performance counters to build ML models, which optimize any memory performance parameter such as throughput or cache miss rate. For new workloads, they use these models and predict the optimal configurations. These techniques are relevant but apply only to the online mapping mode of *FXplore* and we in fact leverage these techniques in Sec. IV-C. In Table III, we compared various ML techniques used in [6] and *FXplore* during the online mapping stage. We observe minimal performance differences among these approaches.

VI. Conclusions

Heterogeneity is a powerful capability in a cluster of servers. It allows workloads to fully exploit the potential of hardware. Unfortunately, commodity servers are expected to handle a diverse range of workloads, which makes it infeasible to customize hardware on these servers such that every workload's performance and energy-efficiency is maximized. In this paper, we demonstrated that firmware options provide relatively strong knobs to introduce soft heterogeneity in a cluster of servers. We showed that both the performance and energy consumption of workloads can be highly sensitive to the firmware settings. However, determining the optimal configuration is not an easy task, because of the exponential complexity in the number of configurations. Thus, we proposed *FXplore* to intelligently explore the firmware configuration design space and reach the optimal configuration with a fast exploration time. Our methodology involves two modes of operation: (1) a one-time offline mode that requires multiple server reboots to explore firmware settings and determine the server sub-clusters and (2) online mapping mode, where incoming workloads from a scheduler get profiled using PMCs and mapped to the sub-clusters wherein a group of servers share the same optimal firmware settings. We demonstrated big gains in exploration time across a range of workloads, which is expected to be substantially higher as the number of options increase in emerging server. We also showed how *FXplore* can identify optimal configurations when co-runners are used. Furthermore, we evaluated the online mapping mode of ML techniques that map new incoming workloads, without requiring a reboot of any server, to existing sub-clusters with pre-determined firmware configurations.

Although some CPU-related hardware settings can be set through the OS and VMMs, many memory and storage related options could usually only be set from firmware. In this paper, we wanted to highlight that there is an opportunity beyond OS tuning to achieve heterogeneity in servers through the firmware. The fact that we can control some of the hardware-software options *via* the OS and VMMs makes our work even more relevant because we can get rid of some of the reboot overheads during the offline mode of *FXplore*. The overall algorithm, however, remains intact even when changing options through the OS or VMMs. Thus, *FXplore* brings us one step closer to realizing a heterogeneous cluster of servers out of an originally homogeneous set of servers with no additional costs or management overheads generally associated with equivalent hardware changes.

Acknowledgments: We thank the anonymous reviewers for their comments. The research of X. Zhan, and S. Reda is supported by NSF grants 1305148 and 1438958.

References

[1] A. Gandhi, M. Harchol-Balter, and R. Das, "Optimal Power Allocation in Server Farms," in *International Conference on Measurement and Modeling of Computer Systems*, pp. 157–168, 2009.

[2] C.-H. Hsu and W. chun Feng, "A power-aware run-time system for high-performance computing," in *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, pp. 1–1, Nov 2005.

[3] J. Liberman and G. Kochhar, "Optimal BIOS settings for high performance compting with poweredge 11g servers," 2010.

[4] J. Beckett, "BIOS performance and power tuning guidelines for dell poweredge 12th generation servers," 2012.

[5] Dell, "Dell poweredge r930 system owner's manual," 2015.

[6] S.-W. Liao, T.-H. Hung, D. Nguyen, C. Chou, C. Tu, and H. Zhou, "Machine learning-based prefetch optimization for data center applications," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pp. 56:1–56:10, 2009.

[7] C. Delimitrou and C. Kozyrakis, "Paragon: QoS-Aware Scheduling for Heterogeneous Datacenters," in *Architectural Support for Programming Languages and Operating Systems*, pp. 77–88, 2013.

[8] J. Mars and L. Tang, "Whare-Map: Heterogeneity in "Homogeneous" Warehouse-Scale Computers," in *International Symposium on Computer Architecture*, pp. 619–630, 2013.

[9] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa, "Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 248–259, 2011.

[10] S. Govindan, J. Liu, A. Kansal, and A. Sivasubramaniam, "Cuanta: Quantifying effects of shared on-chip resource interference for consolidated virtual machines," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, pp. 22:1–22:14, 2011.

[11] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga, "The NAS parallel benchmarks," tech. rep., 1991.

[12] R. Narayanan, B. Ozisikyilmaz, J. Zambreno, G. Memik, and A. Choudhary, "Minebench: A benchmark suite for data mining workloads," in *2006 IEEE International Symposium on Workload Characterization*, pp. 182–188, Oct 2006.

[13] J. Mars, L. Tang, and R. Hundt, "Heterogeneity in "homogeneous" warehouse-scale computers: A performance opportunity," *IEEE Computer Architecture Letter*, vol. 10, pp. 29–32, July 2011.

[14] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, pp. 295–308, 2011.

[15] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi, "An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget," in *International Symposium on Microarchitecture*, pp. 347 –358, 2006.

[16] R. Cochran, C. Hankendi, A. Coskun, and S. Reda, "Pack & Cap: Adaptive DVFS and Thread Packing Under Power Caps," in *ACM/IEEE International Symposium on Microarchitecture*, pp. 175–185, 2011.

[17] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu, "Memory power management via dynamic voltage/frequency scaling," in *Proceedings of the 8th ACM International Conference on Autonomic Computing*, pp. 31–40, 2011.

[18] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini, "Memscale: Active low-power modes for main memory," in *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 225–238, 2011.

[19] D. Lo and C. Kozyrakis, "Dynamic management of turbomode in modern multi-core chips," in *20th IEEE International Symposium on High Performance Computer Architecture*, pp. 603–613, 2014.

[20] T. Leng, R. Ali, J. Hsieh, V. Mashayekhi, and R. Rooholamini, "An empirical study of hyper-threading in high performance computing clusters," in *Linux HPC Revolution*, 2002.

[21] R. E. Grant and A. Afsahi, "Characterization of multithreaded scientific workloads on simultaneous multithreading intel processors," in *In Workshop on Interaction between Operating System and Computer Architecture*, 2005.

[22] S.-w. Liao, T.-H. Hung, D. Nguyen, H. Zhou, C. Chou, and C. Tu, "Prefetch optimizations on large-scale applications via parameter value prediction," in *Proceedings of the 23rd international conference on Supercomputing*, pp. 519–520, ACM, 2009.

[23] N. S. Hussien, S. Sulaiman, and S. M. Shamsuddin, "A review of intelligent methods for pre-fetching in cloud computing environment," in *Recent Advances on Soft Computing and Data Mining*, pp. 647–656, Springer, 2014.