2014 43rd International Conference on Parallel Processing

Modeling the Energy Efficiency of Heterogeneous Clusters

Lavanya Ramapantulu, Bogdan Marius Tudor, Dumitrel Loghin, Trang Vu and Yong Meng Teo
Department of Computer Science
National University of Singapore
13 Computing Drive, Singapore, 117417
Email: {lavanya, teoym}@comp.nus.edu.sg

***Abstract*—Traditional datacenter systems advocate the use of high-performance hardware, resulting in increased power consumption and cooling costs. With increasing availability of systems having diverse performance-to-power ratios, we analyze the energy efficiency of mixing high-performance and low-power nodes in a cluster. Using a model-driven analysis, we predict the heterogeneous mix of nodes that is the most energy-efficient while maintaining a given deadline. Considering service demands of the workloads on cores, memory and I/O devices, we derive Pareto-optimal configurations by matching the execution rate of different nodes. Our mix and match approach determines heterogeneous configurations that exhibit a "sweet region", where energy usage reduces linearly as the deadline is relaxed. Our analysis shows that mixing high-performance and low-power nodes is more energy-efficient than homogeneous datacenter clusters.**

I. Introduction

Energy consumption is a key concern for the industry players that operate some of the world's largest datacenters, such as Google, Facebook, Amazon, among many others [40]. Many research findings propose that low-power processors are an alternative for energy-efficient clusters [18], [20], [23]. On the contrary, other researchers and practitioners indicate that clusters with high-performance nodes are more energy efficient [27], [35], but much remains to be explored to improve the efficiency of scale-out workloads [5].

Datacenter workloads often must obey strict response time constraints. But network conditions, geographical locations and cyclic variation in arrival rates impart different waiting times on jobs even before reaching the datacenter. Many providers favor over-provisioning the servers to shorten the service times of the job to meet the response time deadlines even with such wide variations in waiting time [13], [14]. While a system with only low-power nodes may not service the job fast enough to meet the deadline, a system using only high-performance nodes may require an inordinate amount of energy when operating at higher performance levels than necessary. Ideally, a system should allow a range of configurations that decreases the energy progressively as the deadline is relaxed. This motivates the case for analyzing a heterogeneous cluster system with a mix of high-performance nodes and low-power nodes.

This paper proposes a model-driven analysis of the energy efficiency of executing datacenter workloads on heterogeneous clusters. The objective of the analysis is to determine if a *mix* of high-performance and low-power nodes is more energy-efficient while meeting a given deadline, compared with the case of using only one type of datacenter nodes. To this effect, our idea for minimizing the energy usage is to split the workload in two parts. Both parts are serviced concurrently, the first part executed on high-power nodes and the second part on low-power nodes. But because the two types of nodes have different execution rate, we propose a *matching* technique, that splits the workload such that all high-performance nodes and all the low-power nodes finish the servicing of the job at the same time. By finishing at the same time, the energy incurred by idling in the cluster is minimized. We call this technique *mix and match*. The question of using different types of nodes in datacenter has been addressed in the past [42]. However, the state of the art currently argues that the best approach is to use low-power nodes when the arrival rate of requests is small, and then switch to high-performance nodes when arrival rate grows past a set threshold. In contrast, our *mix and match* technique uses both types of nodes at the same time.

To determine the proportion of workload that is assigned to each type of node, we develop a trace-driven analytical model that determines the energy required to service a job consisting of multiple requests of a scale-out workload. The model considers low-power and high-performance nodes with different Instruction Set Architectures (ISAs). For each type of node, we predict the execution time and energy usage of a job considering the overlap among the response times of service requests to the CPU, the memory and the network I/O devices, as a function of total number of nodes, number of active cores inside each node, and core clock frequency. The model is validated against direct measurements of execution time and energy usage on a heterogeneous cluster with ARM Cortex-A9 and AMD Opteron K10 multicore nodes, for a diverse range of datacenter workloads. We apply our model to determine the execution time and energy usage on different mixes of high-performance AMD and low-power ARM nodes, also varying the core clock frequencies and the number of active cores. Among the large number of configurations, we established the energy-deadline Pareto frontier to obtain a set of heterogeneous configurations which incur the minimum energy needed to execute a given workload within a deadline.

Our analysis is performed by considering a variety of typical datacenter workloads such as web-hosting application *memcached*, multimedia streaming program *x264* from PARSEC, financial analytics program *blackscholes* from PARSEC, real-time speech recognition engine *Julius*, and the openssl implementation of the *RSA-2048 key verification* step of the TLS/SSL encryption mechanism. Specifically, we address the

0190-3918/14 $31.00 © 2014 IEEE
DOI 10.1109/ICPP.2014.41

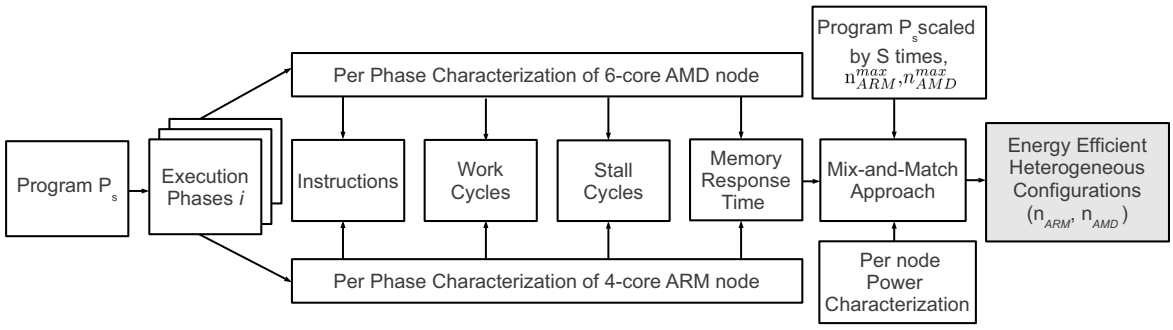

Figure 1: Methodology overview

impact of (i) homogeneity and heterogeneity on the energy used to meet a service time deadline, (ii) the ratio of high-performance to low-power nodes for a given power budget, (iii) the size of the heterogeneous cluster and (iv) job queuing delays on energy efficiency. Our proposed *mix* and *match* approach determines the configurations that significantly reduce the energy progressively (up to 58% on a mix of 16 ARM and 14 AMD nodes) as the deadline is relaxed.

Our key contributions are:

1) We propose a trace-driven modeling approach to determine a mix of high-performance and low-power nodes with different ISAs. This mix minimizes the energy wastage during the service time of a job because matching the execution times among nodes minimizes the idleness in the system. Furthermore, each node operates at the most energy-efficient number of cores and clock frequency.
2) We show that heterogeneous mixes on the energy-deadline Pareto frontier are better than homogeneous configurations. Compared to the solution of switching between a high-performance and a low-power configuration, our approach enables a set of configurations that linearly reduce the energy required to meet a deadline, as the deadline is relaxed.

The rest of the paper is organized as follows. In Section II, we discuss our trace-driven modeling approach. Section III shows the validation of the approach. Section IV presents the energy efficiency analysis. Section V presents related work and Section VI summarizes our approach and analysis.

II. Proposed Model

This section describes our proposed analytical model for the execution time and energy consumption of a cluster with heterogeneous nodes. First, we present the overview of the model and our assumptions. Next, we introduce the derivation of the model.

A. Overview and Assumptions

In this paper, we consider scale-out workloads [12], [26], which are highly parallelizable with negligible inter-node communication. Such programs have repeating parallel phases of execution and has different service demands for the cores, memory and network I/O resources depending on application domain and problem size. We execute such applications on a heterogeneous cluster having n nodes with diverse performance-to-power ratio. All nodes are multicore systems, and all cores inside a node operate at a core clock frequency $f \in [f_{min}, f_{max}]$, where f_{min} and f_{max} are specific to each type of node. Because we target server systems, we consider that the cores inside a node are super-scalar and support out-of-order execution where at least one integer instructions, one floating point instruction and one memory request instruction can be issued within each CPU cycle. Because of the out-of-order architecture, the execution of instructions for which the data is available can be overlapped with the time required to retrieve the data for subsequent instructions [11]. We consider nodes with a single memory controller (i.e. Uniform Memory Architecture) that is equally shared among all the cores of the system. The I/O devices in modern server systems are memory-mapped and can transfer data to and from the main memory with minimal intervention from the CPU, because the transfers are controlled by a specialized processor called a DMA controller. Thus, the activities of the network I/O devices can be completely overlapped with the CPU activities. Most modern multicore systems are covered by this model of execution, including high-performance Intel Xeon or AMD Opteron systems, and low-power ARM Cortex-A8, Cortex-A9, Cortex-A15 and Cortex-A57 systems. In this paper, we consider systems with one I/O network device, and consider that the workloads have negligible storage I/O requirements.

The energy incurred by a node is considered to be split into four components: cores, memory, network I/O device and the rest of the system (disks, GPU, power suply, motherboard circuitry etc.). The power consumption of the cores, memory and network I/O device depends on the activities performed by them. We consider the cores to be in C-state 0, even when they are idling (i.e. cores are never put to sleep, even when they are not executing any workload, which is a common setting for datacenter nodes [7]). The cores can however change the P-state (i.e. cores can change clock frequency). The power consumed by a core depends on its P-state and on the type of compute activity (if a core is executing integer instructions only or floating point instructions only, or both or none). The memory and the network I/O device are considered to have two power states - idle and active. The power consumption incurred by the rest of the system is considered fixed and independent of the workload.

Our model determines the execution time and energy consumed by a program as a function of the number of nodes, number of cores per node and the core clock frequency. By matching the execution times on each node and determining the minimum energy configurations, we obtain the energy-

efficient mix of heterogeneous nodes. This methodology is used to determine a generic mix of heterogeneous nodes. However, for ease of discussion we consider a mix of high-performance AMD x64 nodes and low-power ARM Cortex-A9 nodes, as shown in Table 1. Figure 1 shows our methodology. Given a heterogeneous system with n_{ARM}^{max} and n_{AMD}^{max} nodes that services a job, our approach to derive an energy-efficient heterogeneous mix consists of three main steps.

Node	AMD K10	ARM Cortex-A9
ISA	x86_64	ARMv7-A
Cores/node	6	4
Clock Freq	0.8–2.1 GHz	0.2–1.4 GHz
L1 data cache	64KB / core	32KB / core
L2 cache	512KB / core	1MB / node
L3 cache	6MB / node	NA
Memory	8GB DDR3	1GB LP-DDR2
I/O bandwidth	1Gbps	100Mbps

Table 1: Types of heterogeneous nodes

Notation	Description
$\mathcal{P}$	program
$\mathcal{P}_s$	representative subset of $\mathcal{P}$
W	total work units of $\mathcal{P}$
n	number of nodes
c	number of cores per node
f	clock frequency of a node
Time	
T	total execution time of $\mathcal{P}^*$
T_{CPU}	CPU response time*
$T_{I/O}$	I/O response time*
T_{core}	core response time*
T_{mem}	memory response time*
I_P	total instructions for $\mathcal{P}^*$
I_{P_s}	instructions for ${\mathcal{P}_s}^+$
U_{CPU}	CPU utilization of cores per node$^+$
c_{act}	number of active cores per node$^+$
I_{core}	instructions executed per core*
WPI	work cycles per instruction$^+$
SPI_{mem}	core stall cycles per instruction due to memory$^+$
SPI_{core}	non-memory stall cycles per instruction$^+$
T_{I/O_T}	I/O transfers time*
$\lambda_{I/O}$	I/O requests inter-arrival rate$^+$
T_{act}	time taken by CPU work cycles*
T_{stall}	time taken by CPU stall cycles*
Power	
$P_{CPU,act}$	power of CPU work cycles$^+$
$P_{CPU,stall}$	power of CPU stall cycles$^+$
P_{mem}	power of memory active$^+$
$P_{I/O}$	power of I/O$^+$
P_{idle}	system idle power$^+$
Energy	
E	energy consumed by $\mathcal{P}^*$
E_{CPU}	energy consumed by CPU*
E_{mem}	energy consumed by memory*
$E_{I/O}$	energy consumed by I/O*
E_{idle}	system energy when idle*

Table 2: Model notations

First, using our model, we predict the execution time and energy consumed for all possible configurations of nodes with a given maximum number of nodes for each type. For each configuration, we also compute the workload distribution ratio, to match the execution time among the heterogeneous nodes. Second, we remove the sub-optimal configurations by deriving the Pareto frontier of the energy-time configuration space. Third, given a service time deadline, we output the configuration space(n_{ARM}, n_{AMD}) and the workload distribution that meets this deadline with minimum energy usage. The derivation of Pareto frontier is discussed in section IV-B. We present the modeling of execution time and energy in the following sections using the notations[1] in Table 2.

B. Execution Time Model

This section describes the derivation of the execution time model. Due to the matching technique, the execution time on both types of nodes is the same. Furthermore, the workload is equally distributed among nodes of the same type, thus the execution time of the entire job, T is:

$$T = T_{ARM} = T_{AMD} \tag{1}$$

The job consists of service requests to cores, memory and the I/O device. Due to out-of-order execution and memory-mapped I/O, the response time of these three types of service requests can overlap in time. Thus, simply adding them does not reflect the real service time for the entire job. However, from a measurement point of view, not all the service requests are independent. CPU cores are seen as active by the OS not only when they are servicing core requests (i.e. executing useful work such as integer or floating point instructions), but also when they are just waiting for memory responses. Thus, the CPU time accounts for the response times of both cores and memory. Based on the overlap between CPU time and I/O response time, we define two response times in the system:

1) CPU response time is defined as the total time that a core is executing instructions or waiting for memory requests, accumulated for all the cores in a node.
2) I/O response time is defined as the total time during which any core is waiting for the I/O device.

Because scale-out workloads consist of many repetitions of the same execution phase, either the CPU response time or the I/O response time dominates the total execution time. The faster response time among the two is completely overlapped with the slower one:

$$T_{ARM} = \max(T_{CPU,ARM}, T_{I/O,ARM}) \tag{2}$$

and the same explanation applies to T_{AMD}.

1) CPU Response Time: The CPU response time includes the execution time of the core while performing computations and the stall time of the core while waiting for completion of memory requests. Let T_{core} denote the execution time of the core doing computations and non-memory stalls, and T_{mem} denote the response time of the memory requests. Both ARM and AMD cores support out-of-order executions that may overlap waiting for memory requests with execution of work cycles. Hence the CPU response time is determined by the bottleneck between core and memory.

$$T_{CPU,ARM} = \max(T_{core,ARM}, T_{mem,ARM}) \tag{3}$$

Next, we model how T_{core} and T_{mem} depend on the number of cores and core clock frequency of each node. To do so, we start from the total workload executed by a node.

[1]The listed notations are general. When a parameter is used with a subscript ARM or AMD, it denotes its application to that type of node. The symbol * denotes the model predicted values, while the symbol $^+$ denotes measured values.

At runtime, a unit of workload is translated into a set of machine instructions, which is different among ARM and AMD, because of the different ISAs and micro-architectures. Next, the set of machine instructions is used to model the number of cycles incurred by analyzing the bottlenecks. The number of cycles is modeled as a function of number of cores and core clock frequency of each type of node. In our mix and match approach, the workload is split between the two types of nodes:

$$W = W_{ARM} + W_{AMD} \tag{4}$$

Let $\mathcal{P}_s$ be a representative phase of a scale-out workload. For example, in a video encoding program, the smallest representative phase is the encoding of one frame; in a financial workload, such as pricing of stock options using blackscholes partial differential equations, the smallest representative phase is the computation of the price for one option. Due to different ISAs, the same $\mathcal{P}_s$ translates into different number of instructions on ARM and AMD, and we use $I_{\mathcal{P}_s,ARM}$ to denote the number of machine instructions for ARM ISA that are required to completely execute the phase $\mathcal{P}_s$. The total number of instructions executed by the ARM nodes, I_{ARM}, is:

$$I_{ARM} = W_{ARM} \cdot I_{\mathcal{P}_s,ARM} \tag{5}$$

We measure the number of instructions incurred by $\mathcal{P}_s$, $I_{\mathcal{P}_s}$ on both types of nodes. The instructions executed on each type of node are split among n nodes. Furthermore, within one node the instructions are equally split among the c cores. Depending on the workload, not all c cores may be active during the execution of $\mathcal{P}_s$, due to serialization of the requests on the I/O device. Thus, the average number of cores active in a node, c_{act} is $U_{CPU} \cdot c$. For ARM, the total number of instructions executed by one core inside a node is:

$$I_{core,ARM} = \frac{I_{ARM}}{n_{ARM} \cdot c_{act,ARM}} \tag{6}$$

Next, we derive the execution time per core by modeling the number of cycles. The number of cycles incurred by a core is equal to the work cycles and the stall cycles unrelated to memory accesses:

$$cycles_{core,ARM} = I_{core,ARM} \cdot (WPI_{ARM} + SPI_{core,ARM}) \tag{7}$$

and thus

$$T_{core,ARM} = \frac{cycles_{core,ARM}}{f_{ARM}} \tag{8}$$

As the program scales from $\mathcal{P}_s$ to $\mathcal{P}$, the number of instructions scale, but the ratio of work cycles to instructions remains constant. Similarly the ratio of stall cycles to instructions, SPI_{core}, also remains constant. This hypothesis of constant WPI and SPI_{core} is also validated in Section III-B. Hence, we measure WPI and SPI_{core} for $\mathcal{P}_s$ and then use them to determine the number of cycles for the scale-out program $\mathcal{P}$.

2) Memory Response Time: The CPU response time T_{CPU} shown in Equation 3 is dependent on the memory response time. To determine the memory response time, we measure the stall cycles incurred by the core due to memory requests . If memory is the bottleneck, the total cycles incurred by a core, $cycles_{mem,ARM}$, are the work cycles plus the stall cycles that cannot be overlapped with useful work.

$$cycles_{mem,ARM} = I_{core,ARM} \cdot (WPI_{ARM} + SPI_{mem,ARM}) \tag{9}$$

For the nodes used in this paper, memory operates at a speed independent of the cores and the memory response time is

$$T_{mem,ARM} = \frac{cycles_{mem,ARM}}{f_{ARM}} \tag{10}$$

When the number of cores requesting memory accesses increases, memory response time also increases due to memory contention [36]. This increase in memory response time results in higher CPU stall cycles. According to Equations 9 and 10, memory response time can be obtained by measuring $SPI_{mem,ARM}$ for all values of active cores and core clock frequencies. In Section III-B, we show that $SPI_{mem,ARM}$ regresses linearly over core frequency f_{ARM}.

3) I/O Response Time: Since CPU computation time overlaps with I/O request transfer time, and I/O transfer time in turn overlaps with inter-arrival waiting time of the next request, it suffices to consider the maximum of these two values. Hence, for an I/O bound program the response time is the maximum between the I/O transfer time as a function of I/O bandwidth and the I/O requests inter-arrival rate.

$$T_{I/O,ARM} = \frac{\max(T_{I/O_T,ARM}, \frac{1}{\lambda_{I/O}})}{n_{ARM}} \tag{11}$$

For an I/O bound program, the workload is distributed among the ARM and AMD nodes. As the number of nodes increases for a fixed workload, the I/O response time improves because I/O bandwidth required per node decreases.

C. Energy Model

The energy model determines the total energy consumed by characterizing the power used and the execution time of $\mathcal{P}$. Total energy for a given workload is the sum of the energies consumed within a node by core, memory, I/O and the rest of the system, for all nodes in the cluster.

$$E = E_{ARM} + E_{AMD} \tag{12}$$

$$E_{ARM} = \big(E_{core,ARM} + E_{mem,ARM} + E_{I/O,ARM} + E_{idle,ARM}\big) \times n_{ARM} \tag{13}$$

When the system is completely idle, the power consumption includes the idle power of the cores, memory and I/O devices, as well as the fixed power consumption of the rest of the components. Thus,

$$E_{idle,ARM} = T_{ARM} \cdot P_{idle,ARM} \tag{14}$$

The energy consumed by each core when is active is

$$E_{core,ARM} = ((P_{core,act,ARM} \cdot T_{act,ARM}) + (P_{core,stall,ARM} \cdot T_{stall,ARM})) \cdot c_{act,ARM} \tag{15}$$

where

$$T_{act,ARM} = \frac{I_{core,ARM} \cdot WPI_{ARM}}{f_{ARM}} \tag{16}$$

$$T_{stall,ARM} = \frac{I_{core,ARM} \cdot SPI_{core,ARM}}{f_{ARM}} \tag{17}$$

$I_{core,ARM}$, WPI_{ARM} and $SPI_{core,ARM}$ are obtained as explained in Section II-B1. The energy consumed by the memory and I/O is

$$E_{mem,ARM} = P_{mem,ARM} \cdot T_{mem,ARM} \tag{18}$$

$$E_{I/O,ARM} = P_{I/O,ARM} \cdot T_{I/O,ARM} \tag{19}$$

D. Model Inputs

The trace-driven inputs to our model are obtained from measurements by executing some representative subset of the workloads or micro-benchmarks. We first discuss the measurement of workload dependent input parameters such as CPI, stall cycles followed by the measurement of power parameters.

1) Workload Characterization: Typical scale-out workloads used in datacenters exhibit a lot of parallelism due to both requests and data. The computations of such workloads can be divided into repetitive parallel execution phases within a request and also across a batch of requests [7]. The representative subset $\mathcal{P}_s$ of the scale-out workload used in our model is this repeating parallel phase. For example, in the memcached program, each of the GET, SET and DELETE request types are a parallel phase of execution. We measure the number of instructions, work cycles, stall cycles for a single GET, SET and DELETE command to capture the architecture specific parameters for each type of node. All of these measurements are done using hardware event counters in the respective nodes.

2) Power Characterization: Power for both types of nodes is characterized by the parameters listed in Table 2. During execution, a processor consumes varying amount of power depending on the number of active components. CPU active power, $P_{CPU,act}$, is measured across cores and frequencies for each type of node, using a micro-benchmark that maximizes the CPU utilization. Power incurred by CPU stall cycles, $P_{CPU,stall}$, is measured using a stall micro-benchmark that generates a stream of cache misses to maximize the number of stall cycles. Power used by active memory, P_{mem} for the ARM and AMD node is derived from specifications [1], [24] and I/O power, $P_{I/O}$, is obtained through direct measurement. Idle power of the system, P_{idle}, is measured without any workload. It suffices to do the measurements on a single node of each type, because all the nodes of the same type exhibit very similar power characteristics, which we have validated.

III. Validation

This section shows the validation of our proposed model against measurements of execution time and energy usage for a diverse set of workloads. First, we present the workloads and the system setup. Next we show experimental evidence for our hypothesis of constant WPI and SPI_{core}, and the linearity of SPI_{mem} with core clock frequency. Finally, we summarize the validation results for the predicted execution time and energy across different number of ARM and AMD nodes.

A. Workloads and Setup

Many datacenter workloads must obey strict service time deadlines. To service requests within a deadline, processing is distributed over hundreds of server nodes. Jobs arrive at front-end nodes and are forwarded to a cluster of compute nodes that service job requests. Both response time and the energy incurred by a job are dominated by compute nodes [26]. Thus, we focus on the energy efficiency of compute nodes only. As we are targeting datacenter workloads, we select six programs representing different performance bottlenecks and with different deadline requirements. *EP*, from NPB benchmark [6], is an embarrassingly parallel distributed-memory program that generates random numbers for Monte-Carlo numerical simulation. *Memcached* is widely used by Facebook, Amazon, Twitter, among others, as an in-memory key-value distributed storage. When a key request arrives, a front-end node dispatches the request to a set of nodes that are responsible for storing the key-values belonging to an application. All nodes in the pool perform a key look-up computation, but typically few nodes return the value. However, this operation may exert complex service demands on core, memory and I/O devices [22], [37]. We use memslap running on another system to trigger requests to the memcached server over a 1 Gbps network connection. Note that memslap generates requests with fixed key-value size and uniform popularity. For realistic memcached workload characteristics see [5]. From the PARSEC benchmark suite [8], *x264* represents the widely used encoding algorithm for streaming video, and *blackscholes* represents a quantitative model for determining option pricing. The open source speech recognition engine *Julius* [4] represents the increasing adoption of real-time speech processing workloads originating from smart devices. To analyze the energy efficiency of web security, we use the openssl *RSA-2048* speed benchmark because major web players are increasingly concerned with the in-transit data security and are hardening the https encryptions [3].

To apply the model we perform baseline runs to measure the parameter values with $^+$ symbol in Table 2, for both an ARM and an AMD node. The details of the baseline runs to measure the model inputs are described in Section II-D. We use *perf* to access hardware event counters and to measure execution time, and a Yokogawa WT210 power monitor to measure the power and energy.

B. WPI and SPI_{core}

To validate our hypothesis of constant WPI and SPI_{core} as workload scales from $\mathcal{P}_s$ to $\mathcal{P}$, we use perf to measure the work cycles, non-memory stall cycles and instructions to derive these two model parameters. Figure 2 plots the WPI and SPI_{core} for the EP benchmark with increasing problem sizes from A to C on both ARM and AMD nodes. The plot shows that our hypothesis holds for EP and other programs that we have validated.

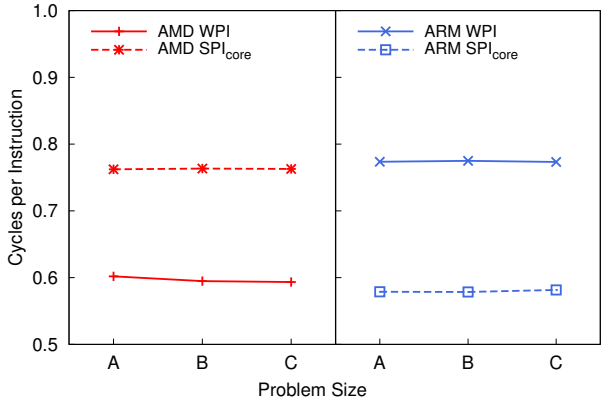

Figure 2: WPI and SPI_{core} across problem size

C. SPI_{mem} Regression over Core Frequency f

To validate our approach, SPI_{mem} is derived by measuring the memory stall cycles and instructions executed across different frequencies and number of cores. Figure 3 shows that as core frequency increases, SPI_{mem} grows linearly. The Pearson's correlation coefficient between SPI_{mem} and f is $r^2 \geq 0.94$, showing very strong linear correlation among them.

Domain	Program	Problem Size	Bottleneck	Execution time error[%]				Energy error[%]			
				AMD node		ARM node		AMD node		ARM node	
				Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
HPC	EP	2,147,483,648 random numbers	CPU	1	5	1	2	6	2	6	5
Web Server	memcached	600,000 GET/SET operations	I/O	7	5	4	8	6	5	5	4
Streaming video	x264	600 frames 704 × 576	Memory	6	4	1	3	9	2	5	8
Financial	blackscholes	500,000 stock options	CPU	1	1	1	1	7	3	5	2
Speech recognition	Julius	2,310,559 samples	CPU	10	2	4	9	7	6	2	5
Web security	RSA-2048	5000 keys verifications	CPU	1	1	10	1	7	2	2	7

Table 3: Single-node validation

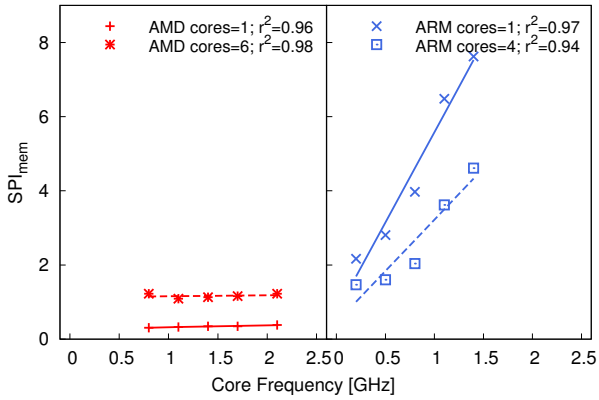

Figure 3: Effect of frequency and number of cores on SPI_{mem}

D. Execution Time and Energy

Model execution time and energy are validated against the measured values, for all workloads described above. We validated two aspects of our mix and match approach. First, we validated the predictions of execution time and energy for one ARM or AMD node across all combinations of number of cores and core clock frequencies. This validation tests the accuracy of selecting the most energy-efficient per-node configuration. Second, we validated the multi-node energy-efficient configurations. Together, these experiments validate our selection of the Pareto-optimal configurations.

Table 3 summarizes the average error and standard deviation on a single ARM or a single AMD node. Table 4 shows the error on a cluster of eight ARM nodes and one AMD node. In summary, the model error is less than 15%, with the main sources of inaccuracy being irregularities among different runs of the same program, and the power characterization.

Program	Configuration		Execution time	Energy
	ARM nodes	AMD nodes	error[%]	error[%]
EP	8	1	3	10
	8	0	3	2
memcached	8	1	10	8
	8	0	3	1
x264	8	1	11	10
	8	0	13	11
blackscholes	8	1	4	7
	8	0	4	13
Julius	8	1	13	1
	8	0	1	2
RSA-2048	8	1	2	8
	8	0	1	12

Table 4: Cluster validation

IV. Energy Efficiency Analysis

For a given service time deadline, this section applies our model to study the energy efficiency of different configurations. Because the energy efficiency of a heterogeneous cluster depends on the energy efficiency of its constituent nodes, we first present the performance-to-power ratio (PPR) of both ARM and AMD nodes. Next, we evaluate if (i) heterogeneous nodes are more energy-efficient than homogeneous systems, (ii) for a given power budget, different mixes of heterogeneous nodes impact energy efficiency, (iii) increasing the number of heterogeneous nodes impacts energy efficiency and (iv) the job queueing delay impacts energy efficiency.

Program	Performance per Watt (PPR)	AMD Node	ARM Node
EP	(random no./s)/W	1,414,922	6,048,057
memcached	(kbytes/s)/W	2,628	5,220
x264	(frames/s)/W	1	0.7
blackscholes	(options/s)/W	2,902	11,413
Julius	(samples/s)/W	21,390	69,654
RSA-2048	(verify/s)/W	9,346	6,877

Table 5: Performance-to-power ratio

A. Performance-to-Power Ratios

PPR is defined as the work done per unit of time, normalized by the average power consumption. This is equivalent to the work done per unit of energy. The PPR computed for the most energy-efficient configuration is shown in Table 5. As observed from the table, ARM has a better PPR than AMD, but with two notable exceptions. For web-security applications such as RSA-2048, AMD has better PPR due to its special instructions that accelerate cryptography processing. X264 encoding algorithm is memory-bound [8], and performs much better on the high memory bandwidth AMD. For the other applications, ARM has a better PPR but lower overall performance. Hence mixing ARM and AMD optimizes both energy efficiency and performance, as shown in Section IV-B.

B. Is Heterogeneity better than Homogeneity?

This section evaluates if heterogeneity reduces energy consumption while still meeting a set deadline. As the total energy depends on the number of active nodes and the number of cores per node and core clock frequency, finding the global optimum configuration is a complex task. For example, a system with ten AMD and ten ARM nodes results in a total of 36,380 possible heterogeneous configurations[2]. An approach to reduce the configuration space is beyond the scope of this paper.

The observations from this section are illustrated using EP and memcached. However, similar observations hold for all

[2]a) Mix of ARM and AMD nodes = 10 (ARM nodes) × 5 (core frequencies per ARM node) × 4 (number of cores per ARM node) × 10 (AMD nodes) × 3 (core frequencies per AMD node) × 6 (cores per AMD node) = 36,000; b) Considering only ARM nodes, 10 × 5 × 4 = 200; c) Considering only AMD nodes, 10 × 3 × 6 = 180. Total $= 36,000 + 200 + 180 = 36,380$

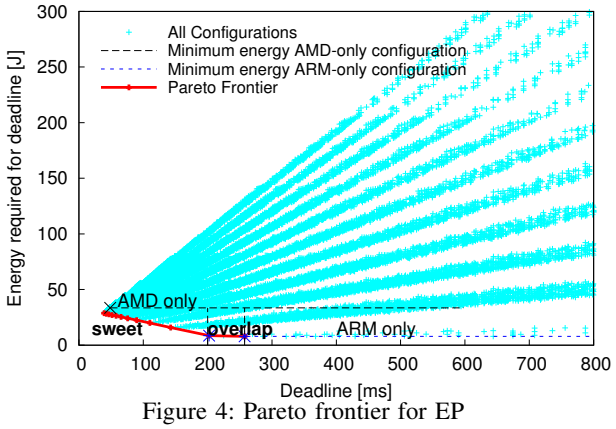

Figure 4: Pareto frontier for EP

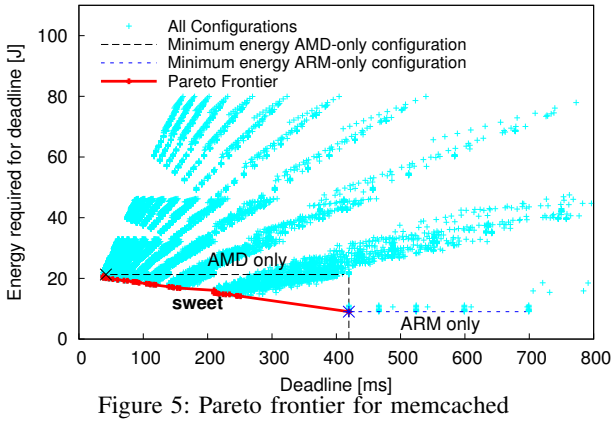

Figure 5: Pareto frontier for memcached

workloads where the PPR of ARM is better than that of AMD. For our energy efficiency analysis we use 50,000 memcached requests as one job. For easy comparison across workloads, we choose an input of 50 million random numbers for EP, so that the execution time is roughly similar to memcached. However, we note that the input size does not impact the conclusion of the analysis because increasing the input size leads to linear increase in both execution time and energy usage.

Figures 4 and 5 plot the energy incurred to finish a job for all possible configurations for memcached and EP respectively. Each point in this plot represents a different configuration where a configuration is determined by the number of ARM and AMD nodes, number of cores per node and the core clock frequency. For each configuration point, the x-axis denotes the job service time and the y-axis represents the corresponding energy used. Given a deadline, there exist a set of configurations that meet this deadline. The configuration that meets the deadline with the minimum energy usage is *Pareto optimal*. The set of all Pareto optimal points across all possible deadlines forms the energy-deadline *Pareto frontier*.

In Figure 4, we show the minimum energy incurred by AMD-only and ARM-only configurations. The thicker line represent the Pareto frontier that partially overlaps with ARM-only minimum energy configurations. Because of the overlap, the Pareto frontier can be divided into two parts. The left part consists of mixes of AMD and ARM nodes, and represents a "sweet region"[3] where relaxing the deadline linearly reduces the energy used. The sweet region is bounded by the energy incurred by the homogeneous configurations, with ARM representing the lower bound and AMD the upper bound. The right part, represents an "overlap region" consisting of ARM-only configurations. The overlap region exists when the program is compute-bound. For such a program, decreasing the number of cores or the core clock frequency increases the execution time and decreases the energy. Figure 4 shows that, in the overlap region, as the deadline is relaxed from 200ms to 250ms, a decrease in energy usage occurs because of reducing the number of cores and the core clock frequency. However, for I/O bounded executions, improving performance is only possible by increasing the number of nodes. Thus, I/O bounded programs do not exhibit an overlap region as shown in Figure 5 where the energy incurred by memcached on homogeneous systems is constant even as deadline is relaxed.

Observation 1: *Heterogeneity allows larger energy savings compared to homogeneous systems while maintaining the same service time deadline.*

This analysis may seem unfair because we compare 10 AMD nodes with 10 AMD plus 10 ARM. In the next section, we show that this observation holds even when we replace some AMD nodes with ARM nodes. The question then becomes how many AMD nodes should be replaced by ARM nodes such that we improve the energy efficiency of meeting a deadline.

C. What is a Good Mix of High-performance to Low-power Nodes?

Since datacenters often have an upper bound on their peak power consumption, we consider a fixed peak power budget drawn by our system that constrains the maximum number of nodes. Based on peak power proportion between ARM and AMD nodes, we analyze the impact of replacing some high-performance AMD nodes by low-power ARM nodes such that the total peak power is within the budget.

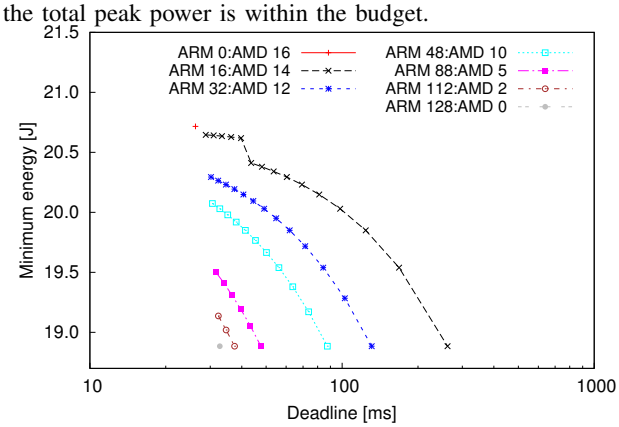

Figure 6: Heterogeneous mixes for memcached

Figures 6 and 7 show[4] the impact of changing the number of ARM and AMD nodes, for a given budget of 1kW. We use an ARM to AMD power substitution ratio[5] of 8:1. The graphs

[3] A sweet region is a union of Pareto optimal heterogeneous sweet spots.

[4] Henceforth, each figure plots Pareto frontiers with x-axis in log-scale.

[5] Since each AMD node draws a peak power of 60W and each ARM node draws a peak power of 5W, one AMD node can be replaced by 12 ARM nodes. Factoring the 20W peak power drawn by the switch [2] that connects the ARM nodes, gives us a power substitution ratio of 8:1.

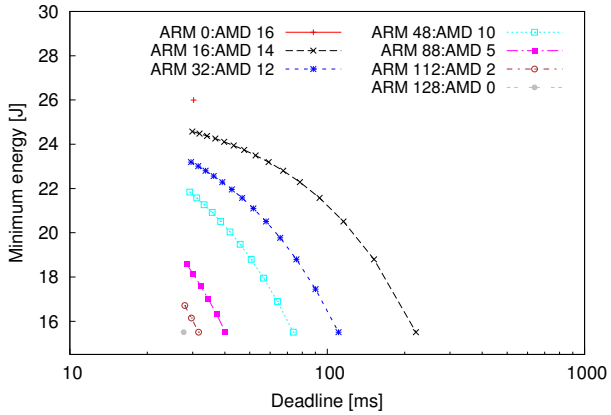

Figure 7: Heterogeneous mixes for EP

clearly show that heterogeneous mixes with a larger number of ARM nodes incur lower energy for a given execution time.

Observation 2: *Replacing even a few high-performance nodes based on the power substitution ratio, introduces a sweet region.*

However, using only low-power nodes may not meet the service time deadline. For example, Figure 6 shows that low-power ARM only configurations do not meet deadlines smaller than 30ms. For an application that is compute-bound, such as EP, replacing even a few AMD nodes triggers a sweet region. However, the most energy-efficient configuration is achieved by replacing all AMD nodes with ARM nodes. This is possible because, while eight ARM nodes are power-equivalent to one AMD nodes, the execution rate of eight ARM nodes is higher than one AMD node.

D. Are Larger Mixes of Heterogeneous Nodes Better?

Using the same power substitution ratio, Figures 8 and 9 show that increasing the number of heterogeneous nodes does not change the energy bounds of a sweet region. Secondly, it increases the number of configurations on a sweet region. Thirdly, as expected, increasing the number of nodes results

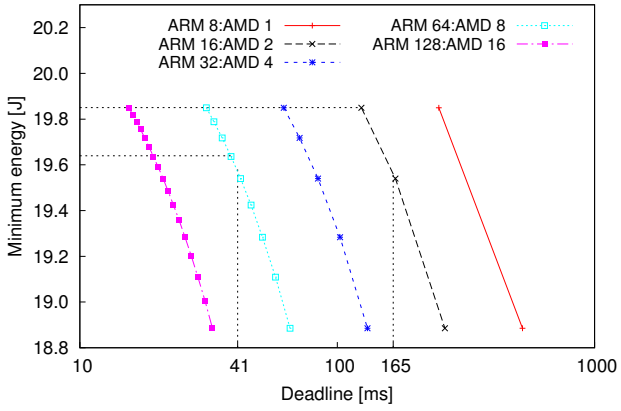

Figure 8: Increasing cluster size for memcached

in faster execution time, causing the sweet regions to shift to the left.

Observation 3: *Increasing the number of nodes in a heterogeneous mix, while maintaining the same power substitution ratio increases the number of configurations on a sweet region without changing its energy bounds.*

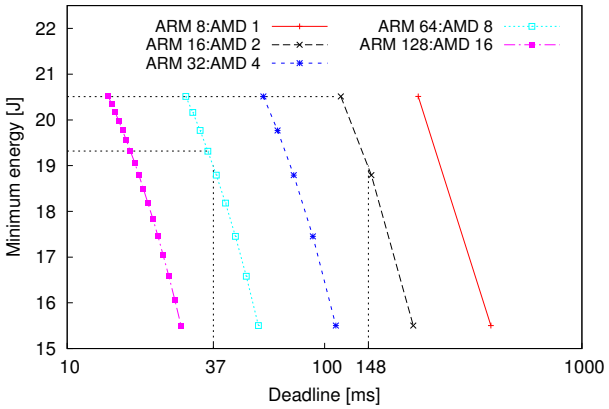

Figure 9: Increasing cluster size for EP

This observation has an interesting implication. Given n jobs, it is better to schedule both on the same cluster than assigning each job to n clusters each with $\frac{1}{n}$ of the total capacity. For example, consider four memcached jobs with a deadline of 165 milliseconds each, and 64 ARM and 8 AMD nodes, and two possible setups: (i) we can create four clusters, each with 16 ARM and 2 AMD nodes, or (ii) one large cluster with all nodes. In the first setup, Figure 8 indicates that the configuration that meets the deadline incurs 19.8 Joules per job. In contrast, the configuration that meets a deadline of four times smaller (41 ms), incurs 19.6 Joules per job.

E. Impact of Jobs Queueing Delay

So far we assumed that each job does not wait for other jobs inside a datacenter. Next, we extend the Pareto frontier to model job arrivals with waiting time.

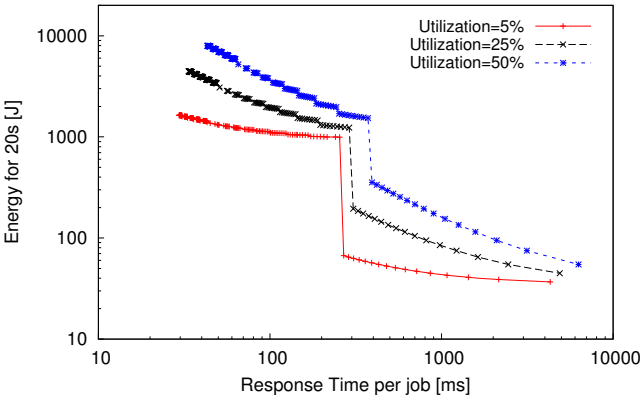

Figure 10: Effect of job queueing delay on cluster utilization

We model the arrivals and departures of jobs to a datacenter using a M/D/1 queueing model. Jobs are assumed to arrive with inter-arrival time exponentially distributed with parameter λ_{job}, and are queued in a dispatcher node until all the previous jobs have been serviced. The service time for a job is considered fixed, and modeled by our *matching* scheduling policy. According to the M/D/1 queueing model, the utilization of the cluster is $U = T\lambda_{job}$, where T is the service time.

We analyze the effect of changing arrival rate by varying the arrival rate such that the utilization varies between 0 and 1. Figure 10 plots in log-log scale the total energy consumed by a cluster of 16 ARM and 14 AMD nodes servicing multiple

memcached jobs each with 50,000 requests, for an observation period of 20 seconds. We plot three profiles of utilization, corresponding to a tenfold increase in arrival rate. For a configuration point that does not use all 16 ARM and 14 AMD nodes, we consider the unused nodes as turned off. As arrival rate increases, the average waiting time in the dispatcher queue also increases. To meet the same response time deadline, jobs need to be serviced faster, which requires a configuration with more high-performance nodes. Thus, as the utilization increases from 5% to 50%, the energy required to meet the same deadline increases almost by an order of magnitude.

However, Figure 10 shows that the sweet region is still present, for all values of utilization. Unlike our previous analysis where we considered only energy incurred by job service time, the sweet region has a more complex shape and can be divided into two linear regions delimited by a sharp drop in the energy used. In the leftmost part of the sweet region, the configurations always include high-performance AMD nodes. Because AMD idle power is 45 watts, the idle energy use is considerable. In contrast, the rightmost part of the sweet region consists of configurations with only ARM nodes, which idle at less than 2 watts, thus incurring much lower idle energy.

When considering the idle energy and job queueing delay of a system, the energy reductions achievable by heterogeneous systems are much larger, spanning almost two orders of magnitude. As cluster utilization increases due to faster job arrivals, the energy savings are further amplified, but the minimal response time achievable is reduced.

Observation 4: *Energy savings achieved by the mix and match approach are amplified when cluster utilization increases.*

V. Related Work

Previous work on energy efficiency of heterogeneous clusters can be classified into: (i) level of heterogeneity including chip level, node level or cluster level, (ii) energy efficiency approaches, and (iii) scheduling of jobs in cluster. Each of these are discussed and compared with our work.

1) Level of Heterogeneity: At chip-level, Van Craeynest and Eeckhout propose a study of single ISA multicore heterogeneity to understand the extent to which both system-level throughput and per-program performance can be simultaneously satisfied [38]. Dynamic core heterogeneity is investigated for speeding up programs with sequential and parallel fractions [17] and programs that alternate between regions with high thread-level parallelism and instruction-level parallelism [31]. More recently, with the emergence of ARM big.LITTLE, a hierarchical power management approach to optimize the performance per watt within a thermal-design power budget [29]. At node-level, KnightShift tightly couples a single high-performance server with a low-power server to enable two energy-efficient operating modes [42]. Mixing CPU and accelerators is another method used to improve the efficiency per watt at node-level [41]. At a cluster level, WhareMap [25] explores performance improvements by using existing heterogeneity in modern warehouse scale computers, while we model and analyze the impact of heterogeneity because of diverse performance-to-power ratios. Chun et al. [10] study the feasibility and potential of hybrid datacenters with Xeon and Atom platforms, but considering only one node of each type, while we explore heterogeneous mixes of several nodes. While Nathuji et al. [30] exploit across-platform heterogeneity for power efficiency, they do not consider nodes with diverse performance-to-power ratios. Moreover, their approach uses throughput as a measure of performance, while we model the execution time.

Closer to our approach is Heath et al. [16] who propose a modeling technique to optimize energy for a cluster of nodes with different CPUs and network capabilities. The model considers request distribution among nodes to balance resource utilization. In contrast, we analyze energy efficiency by matching the execution rates among all the nodes in the cluster. Central to our approach is the existence of two types of nodes, with significant differences in power consumption. Furthermore, our approach uses various mixes of high-performance and low-power nodes to achieve linear reductions of energy used as the execution time deadline is relaxed.

2) Energy Efficiency: There are many techniques to improve energy efficiency in cluster systems. Dynamic voltage scaling to mitigate pipeline imbalances within a core are proposed by [21]. PEPON [32] discusses power distribution among multiple-cores to maximize performance without exceeding a given power budget. Algorithms for dynamic power management of clusters are discussed in [9], [19], [28], These techniques complement our approach as we do not propose dynamic power management techniques within a core or node. We analyze the improvements in energy reductions for a given power budget but with a heterogeneous mix of *both* high-performance and low-power CPU nodes. While Guevara et al. [15] consider performance-efficiency trade-offs of heterogeneous processors, their approach uses proxies to compose bids on behalf of applications for specific type of hardware. On the other hand, our model determines energy efficient configurations for a *single workload* on a given set of heterogeneous mix of nodes. Our previous work uses a modeling approach to determine a configuration of cores and core clock frequency that optimizes energy within a single low-power ARM node [37]. However, the modeling approach in this paper is constructed to analyze the Pareto-optimal energy-deadline space of clusters with both low-power and high-performance nodes.

3) Workload Scheduling: Hybrid-MR [33] discusses algorithms for dynamic resource management of workloads. PIE [39] dynamically adjusts the scheduling of jobs for single-ISA heterogeneous multicores. However, our approach distributes a single workload equally among nodes of the same type and matches the execution time among heterogeneous nodes to achieve energy efficiency for a given deadline. While Spicuglia et al. [34] present an algorithm that balances requests from multiple applications on heterogeneous servers and optimizing the mix of multiple applications on a single server, our work models a single workload on heterogeneous clusters.

VI. Conclusions

This paper proposes a model-driven analysis to determine an energy-efficient mix of high-performance and low-power nodes in a cluster. By modeling the workload service demands on cores, memory and I/O devices of a node, we derive the energy-efficient *mix* of nodes that services a job while maintaining a service time deadline. We obtain a Pareto-optimal set of configurations by *matching* the execution time of different nodes to minimize system idle time. This *mix and match* approach exposes a "sweet region" containing a set of

configurations where the energy used by a job reduces linearly as its service time deadline is relaxed. Unlike homogeneous systems that do not exhibit a sweet region, heterogeneous configurations generally use less energy to meet the same service time deadline. A sweet region can be obtained by replacing even a few high-performance nodes with low-power nodes while maintaining a power budget. To analyze different heterogeneous mixes, we characterize workloads representing two extreme points of the datacenter computing spectrum on a cluster of AMD Opteron and ARM Cortex-A9 nodes. We observe that changing from a homogeneous AMD cluster to a heterogeneous AMD and ARM cluster reduces energy by up to 44% for memcached and 58% for EP, while maintaining the same execution time deadline.

ACKNOWLEDGEMENTS

This work was in part supported by the National Research Foundation, Prime Ministers Office, Singapore under its Competitive Research Programme (CRP Award No. NRF-CRP8-2011-08).

REFERENCES

[1] DDR3 Specification, http://www.webcitation.org/6JN7G4r3x, 2010.

[2] Cisco Catalyst 2960-S and 2960 Series Switches, http://www.webcitation.org/6JPSUNMj0, 2011.

[3] Google finishes 2048-bit RSA migration, Yahoo to encrypt all data early next year, http://www.webcitation.org/6LaPMkEj0, 2013.

[4] Julius, http://julius.sourceforge.jp/, 2013.

[5] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, M. Paleczny, Workload Analysis of a Large-scale Key-value Store, *Proc. of ACM SIGMETRICS/PERFORMANCE*, pages 53–64, 2012.

[6] D. Bailey, T. Harris, W. Saphir, R. Van Der Wijngaart, A. Woo, M. Yarrow, The NAS Parallel Benchmarks 2.0, Technical Report NAS-95-020, NASA Ames Research Center, 1995.

[7] L. A. Barroso, J. Clidaras, U. Hlzle, The datacenter as a computer: An introduction to the design of warehouse-scale machines, second edition, *Synthesis Lectures on Computer Architecture*, 8(3):1–154, 2013.

[8] C. Bienia, S. Kumar, J. P. Singh, K. Li, The PARSEC Benchmark Suite: Characterization and Architectural Implications, *Proc. of PACT*, pages 72–81, 2008.

[9] J.-J. Chen, K. Huang, L. Thiele, Power Management Schemes for Heterogeneous Clusters under Quality of Service Requirements, *Proc. of 26th ACM Symposium on Applied Computing*, pages 546–553, 2011.

[10] B.-G. Chun, G. Iannaccone, G. Iannaccone, R. Katz, G. Lee, L. Niccolini, An Energy Case for Hybrid Datacenters, *SIGOPS Operating Systems Review*, 44(1):76–80, Mar. 2010.

[11] J. Corbet, A. Rubini, G. Kroah-Hartman, *Linux Device Drivers, 3rd Edition*, O'Reilly Media, Inc., 2005.

[12] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, B. Falsafi, Quantifying the Mismatch between Emerging Scale-Out Applications and Modern Processors, *ACM TOCS*, 30(4):15:1–15:24,2012.

[13] A. Gandhi, M. Harchol-Balter, R. Raghunathan, M. A. Kozuch, AutoScale: Dynamic, Robust Capacity Management for Multi-Tier Data Centers, *ACM TOCS*, 30(4):14:1–14:26, Nov. 2012.

[14] A. Greenberg, J. Hamilton, D. A. Maltz, P. Patel, The Cost of a Cloud: Research Problems in Data Center Networks, *SIGCOMM Computer Communication Review*, 39(1):68–73, Dec. 2008.

[15] M. Guevara, B. Lubin, B. C. Lee, Navigating Heterogeneous Processors with Market Mechanisms, *Proc. of HPCA*, pages 95–106, 2013.

[16] T. Heath, B. Diniz, E. V. Carrera, W. Meira, Jr., R. Bianchini, Energy Conservation in Heterogeneous Server Clusters, *Proc. of PPoPP*, pages 186–195, 2005.

[17] M. Hill, M. Marty, Amdahl's Law in the Multicore Era, *Computer*, 41(7):33–38, 2008.

[18] V. Janapa Reddi, B. C. Lee, T. Chilimbi, K. Vaid, Web Search Using Mobile Cores: Quantifying and Mitigating the Price of Efficiency, *Proc. of ISCA*, pages 314–325, 2010.

[19] U. R. Karpuzcu, A. Sinkar, N. S. Kim, J. Torrellas, EnergySmart: Toward energy-efficient manycores for Near-Threshold Computing, *Proc. of HPCA*, pages 542–553, 2013.

[20] L. Keys, S. Rivoire, J. D. Davis, The Search for Energy-efficient Building Blocks for the Data Center, *Proc. of ISCA*, pages 172–182, 2010.

[21] S. Lee, S. Das, T. Pham, T. Austin, D. Blaauw, T. Mudge, Reducing Pipeline Energy Demands with Local DVS and Dynamic Retiming, *Proc. of ISLPED*, pages 319–324, 2004.

[22] K. Lim, D. Meisner, A. G. Saidi, P. Ranganathan, T. F. Wenisch, Thin Servers with Smart Pipes: Designing SoC Accelerators for Memcached, *Proc. of ISCA*, pages 36–47, 2013.

[23] K. Lim, P. Ranganathan, J. Chang, C. Patel, T. Mudge, S. Reinhardt, Understanding and Designing New Server Architectures for Emerging Warehouse-Computing Environments, *Proc. of ISCA*, pages 315–326, 2008.

[24] K. T. Malladi, B. C. Lee, F. A. Nothaft, C. Kozyrakis, K. Periyathambi, M. Horowitz, Towards Energy-proportional Datacenter Memory with Mobile DRAM, *Proc. of ISCA*, pages 37–48, 2012.

[25] J. Mars, L. Tang, Whare-map: Heterogeneity in "Homogeneous" Warehouse-scale Computers, *Proc. of ISCA*, pages 619–630, 2013.

[26] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, T. F. Wenisch, Power Management of Online Data-intensive Services, *SIGARCH Computer Architecture News*, 39(3):319–330, 2011.

[27] D. Meisner, T. F. Wenisch, Does Low-power Design Imply Energy Efficiency for Data Centers?, *Proc. of ISLPED*, pages 109–114, 2011.

[28] K. Meng, R. Joseph, R. P. Dick, L. Shang, Multi-optimization Power Management for Chip Multiprocessors, *Proc. of PACT*, pages 177–186, 2008.

[29] T. S. Muthukaruppan, M. Pricopi, V. Venkataramani, T. Mitra, S. Vishin, Hierarchical Power Management for Asymmetric Multi-core in Dark Silicon Era, *Proc. of DAC*, pages 174:1–174:9, 2013.

[30] R. Nathuji, C. Isci, E. Gorbatov, Exploiting Platform Heterogeneity for Power Efficient Data Centers, *Proc. of ICAC*, pages 5–5, 2007.

[31] M. Pricopi, T. Mitra, Bahurupi: A Polymorphic Heterogeneous Multi-core Architecture, *ACM TACO*, 8(4):22:1–22:21, 2012.

[32] A. Sharifi, A. K. Mishra, S. Srikantaiah, M. Kandemir, C. R. Das, PEPON: Performance-aware Hierarchical Power Budgeting for NoC based Multicores, *Proc. of PACT*, pages 65–74, 2012.

[33] B. Sharma, T. Wood, C. R. Das, HybridMR: A Hierarchical MapReduce Scheduler for Hybrid Data Centers, *Proc. of ICDCS*, 2013.

[34] S. Spicuglia, M. Björkqvist, L. Y. Chen, G. Serazzi, W. Binder, E. Smirni, On Load Balancing: A Mix-aware Algorithm for Heterogeneous Systems, *Proc. of ICPE*, pages 71–76, 2013.

[35] D. Tsirogiannis, S. Harizopoulos, M. A. Shah, Analyzing the Energy Efficiency of a Database Server, *Proc. of SIGMOD*, pages 231–242, 2010.

[36] B. Tudor, Y.-M. Teo, S. See, Understanding off-chip memory contention of parallel programs in multicore systems, *Proc. of ICPP*, pages 602–611, Sept 2011.

[37] B. M. Tudor, Y. M. Teo, On Understanding the Energy Consumption of ARM-based Multicore Servers, *Proc. of SIGMETRICS*, pages 267–278, 2013.

[38] K. Van Craeynest, L. Eeckhout, Understanding Fundamental Design Choices in Single-ISA Heterogeneous Multicore Architectures, *ACM TACO*, 9(4):32:1–32:23, 2013.

[39] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, J. Emer, Scheduling Heterogeneous Multi-cores through Performance Impact Estimation (PIE), *Proc. of ISCA*, pages 213–224, 2012.

[40] A. Vasan, A. Sivasubramaniam, V. Shimpi, T. Sivabalan, R. Subbiah, Worth their watts? - an empirical study of datacenter servers, *Proc. of HPCA*, pages 1–10, Jan 2010.

[41] J. Wang, N. Rubin, H. Wu, S. Yalamanchili, Accelerating Simulation of Agent-Based Models on Heterogeneous Architectures, *Proc. of GPGPU*, 2013.

[42] D. Wong, M. Annavaram, KnightShift: Scaling the Energy Proportionality Wall through Server-Level Heterogeneity, *Proc. of MICRO*, pages 119–130, 2012.