Optimal Scheduling for Asymmetric Multi-core Server Processors

by

Bharathwaj Raghunathan

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2014

Author's Declaration

This thesis consists of material all of which I authored or co-authored [1, 2]: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

In Chapter 1:

Contributor	Contributions
Raghunathan, B.	Manuscript writing
Garg, S.	Manuscript editing and Figure 1.1

In Chapter 3:

Contributor	Contributions
Raghunathan, B.	Manuscript writing
Garg, S.	Manuscript editing and Figure 3.2

Abstract

The arrival rate of jobs at servers in a data-center can vary significantly over time. The servers in data-centers are typically multi-core processors, which allow jobs to be processed at different degrees of parallelism (DoPs), i.e., the number of threads spawned by a job. In this thesis, we show analytically as well as empirically that the DoP which minimizes the service time of jobs varies with the arrival rate of jobs. Also, recent trends have shown a move towards asymmetric multi-core server processors. These processors are made up of multiple clusters, each consisting of cores of different type and of which only one cluster can be turned on at a given point in time while the others remain "dark". We show that the choice of the optimal cluster is dependent on the arrival rate. Based on these observations, we propose a run-time scheduler that determines the optimal DoP and performs inter-cluster migration to minimize the mean total service time. The main contributions of this thesis are

- We propose a queueing theoretic model to determine the mean service time of jobs as function of the DoP, number of parallel jobs and the cluster choice.
- Based on the queueing theoretic model, we show that both the optimal DoP and cluster choice are dependent on the job arrival rate, and propose a run-time scheduler that makes optimal optimal cluster migration and DoP selection decisions to minimize mean service time.

Acknowledgements

Firstly, I thank my supervisor Prof. Siddharth Garg for his help and guidance throughout my stay at the UWaterloo. I have learnt so much from him on how to perform good research.

I thank my thesis readers, Prof. Hiren D. Patel and Prof. Mahesh Tripunitara for their helpful feedback on this thesis and for being wonderful instructors.

I am gratefully to the UWaterloo community for providing such a great environment which has helped me grow as a person.

Finally, I thank MITACS for their financial support during my time at UWaterloo.

Dedication

I dedicate this thesis to my parents and my grandmother. They have always been there for me.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In recent times, a new paradigm shift in computing has emerged. This paradigm shift termed "Cloud Computing" is the availability of computing resources over large networks such as the internet. A number of companies like Google, Microsoft, and Amazon deliver a variety of services like search and e-commerce through the cloud. These services are hosted in warehouse scale data-centers.

Data-centers are composed of a network of servers built using commodity or enterprise class multi-core processors. The rate at which jobs arrive at the data-center for processing, i.e. the ***job arrival rate***, can change significantly in a matter of hours or even minutes. As an example, analysis of Wikipedia data shows up to 4× variation in article update arrival rates over the course of a day, averaged over one minute intervals [5]. These jobs are assigned to individual servers by a data-center scheduling mechanism. Hence, variations in job arrival rate at the data-center level translate to variations in the arrival rate at the server level. Barroso et al. [6] have noted that server utilization in production data-centers varies widely.

The availability of multiple cores on server processors enables the parallel execution of data-center jobs. The ***degree of parallelism*** (DoP) of a job refers to the number of parallel threads used to execute that job. As jobs submitted queue up for processing, a run-time scheduler decides the number of jobs to execute in parallel (J), and the DoP of each job (D). For example, given a search workload, the scheduler can execute the the workload in sequence, while parallelizing each job to the greatest possible degree. In this scenario, the DoP of each job would be $D = N_c$, where N_c is equal to the number of cores on the processor, and $J = 1$ since jobs are executing in sequence. At the other extreme, the scheduler can execute $J = N_c$ search queries in parallel, with each search query running on

a single core (i.e., the DoP $D = 1$). In general, any combination of J and D that satisfies $J \times D = N_c$ can be chosen.

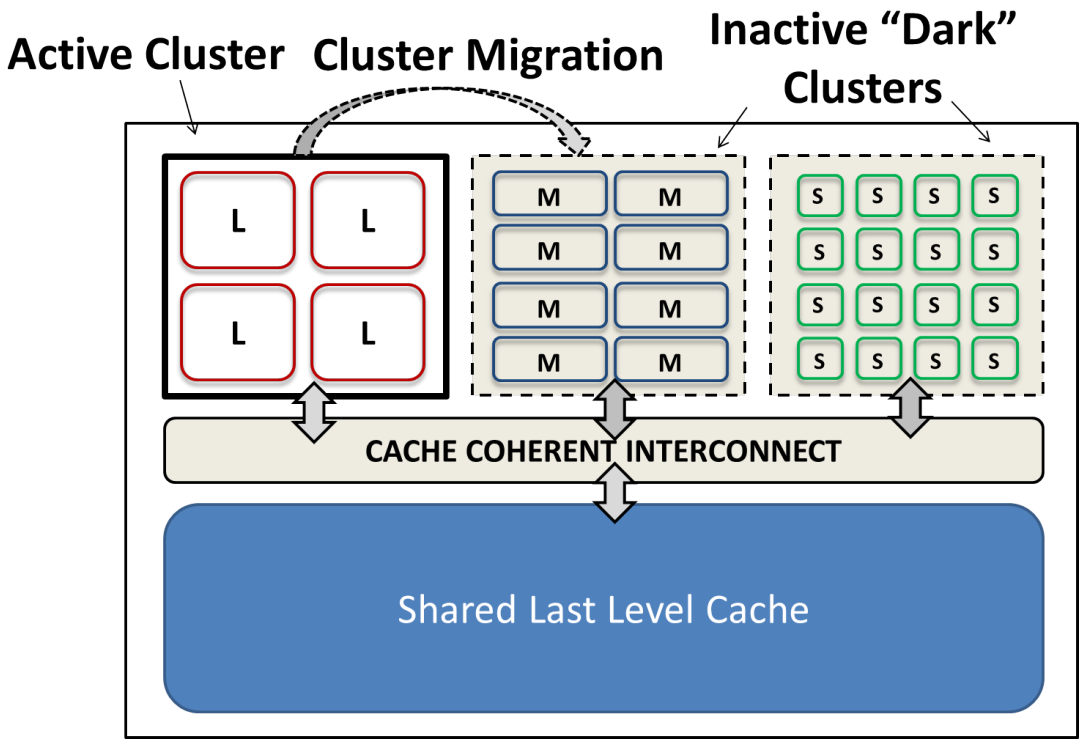


Figure 1.1: A clustered asymmetric multi-core processor with three clusters.

The abundance of transistors on a chip has motivated a move towards ***asymmetric multi-core*** processor designs. This thesis focuses on asymmetric multi-core processors that contain multiple clusters of cores. An example of the same is the recently introduced Exynos processor [7] based on the ARM big.LITTLE architecture [4]. Figure 1.1 provides an example of such a processor that consists of three clusters. Each cluster has many cores of the same type, but heterogeneity exists across clusters. In the figure, three clusters have small (S), medium (M), and large (L) cores; small cores provide lower performance than medium and large cores but also consume less power.

Asymmetric multi-core processors are over-provisioned, i.e., all clusters can not be switched on at the same time; doing so would violate the chip power budget. This is a consequence of a trend towards so called dark silicon which is the part of the chip that must be left unpowered due to power constraints. As a result, usually only one cluster is

active at any point in time, while the others remain dark. Jobs can be migrated between clusters as and when required. This is referred to as cluster migration mode [7]. In this work we assume that each cluster will have a sufficient number of cores to consume the entire chip's power budget when active. We note that the clustered processors available today do not have this property. For example the Exynos processor has the same number of little cores as big cores (hence the little cluster consumes less power)

In the context of asymmetric multi-core architectures, an important challenge is the design of run-time schedulers that minimize mean service time within a given power budget. In particular, the run-time scheduler must determine: (a) which cluster to execute jobs on (and appropriately migrate jobs to the selected cluster), and (b) the optimal DoP (D) for each job on that cluster so as to minimize the mean service time of jobs.

The *goal* of this thesis is to determine optimal run-time scheduling policies to minimize the mean service time of jobs for asymmetric multi-core processors in the presence of varying job arrival rates. To facilitate this, we propose a queueing theoretic model to help determine the mean service time of jobs.

1.1 Thesis Contributions

The main contribution of this thesis are:

- We propose a queueing theoretic model to determine the mean service time of jobs as a function of the DoP (D), number of parallel jobs (J) and the cluster choice. The model is extensively validated against detailed simulations on a micro-architectural simulator and shown to be accurate over a wide range of job arrival rates.
- Based on the queueing theoretic model, we show that both the optimal DoP and cluster choice are dependent on the job arrival rate, and propose a run-time scheduler that makes optimal cluster migration and DoP selection decisions to minimize mean service time.
- Our empirical analysis using real job arrival rate curves from production data-centers show significant improvements in mean service time using the proposed job arrival rate aware scheduler over conventional schedulers that only statically optimize the DoP and cluster choice.

Much of the existing work on heterogeneous or asymmetric processors is motivated by heterogeneity in job/application characteristics. On the other hand, an interesting

observation in our work is that *even if* all the jobs are identical, for example, even if the server is only executing jobs of the same type, variations in the job arrival rate still motivate the design of asymmetric processors with different core types.

1.2 Outline

Chapter 2 talks about prior work on DoP optimization, asymmetric clustered architectures, and various thread scheduling policies. Then we move on to discussing the proposed approach and queueing theoretical model in Chapter 3. Chapter 4 describes the experimental setup and Chapter 5 discusses the experimental results and the evaluations of the proposed approach. Finally, Chapter 6 provides a conclusion to this thesis and Chapter 7 looks at possible extensions to the thesis.

Chapter 2

Background and Related Work

In this chapter, we talk in more detail about cloud computing and dark silicon. Also, we discuss prior work in detail.

2.1 Cloud Computing

Cloud computing, a recent trend in computing, is the availability of computing resources over large networks like the Internet. A number of companies like Google, Amazon, IBM, and Microsoft provide a variety of services on the cloud. Some of these service include Software-as-a-service (SaaS), Platform-as-a-service (PaaS), Infrastructure-as-a-service (IaaS) to name a few. Our work deals with the SaaS model, where users submit jobs to a centrally hosted server.

Cloud computing services are hosted in warehouse scale data-centers. Companies such as Google and Facebook have multiple data-centers; many data-centers use as much electricity as small town and produce enough heat to provide heating for other facilities. Data centers contain thousands of servers made from enterprise or commodity class multi-core processors. The rate at which jobs arrive at the data-center for processing, i.e. the ***job arrival rate***, can change significantly in a matter of hours or even minutes. Figure 2.1 shows a graph that traces the arrival rate variations in a 24 hour period obtained from a Facebook map-reduce cluster.[1]

[1] https://github.com/SWIMProjectUCB/SWIM/wiki

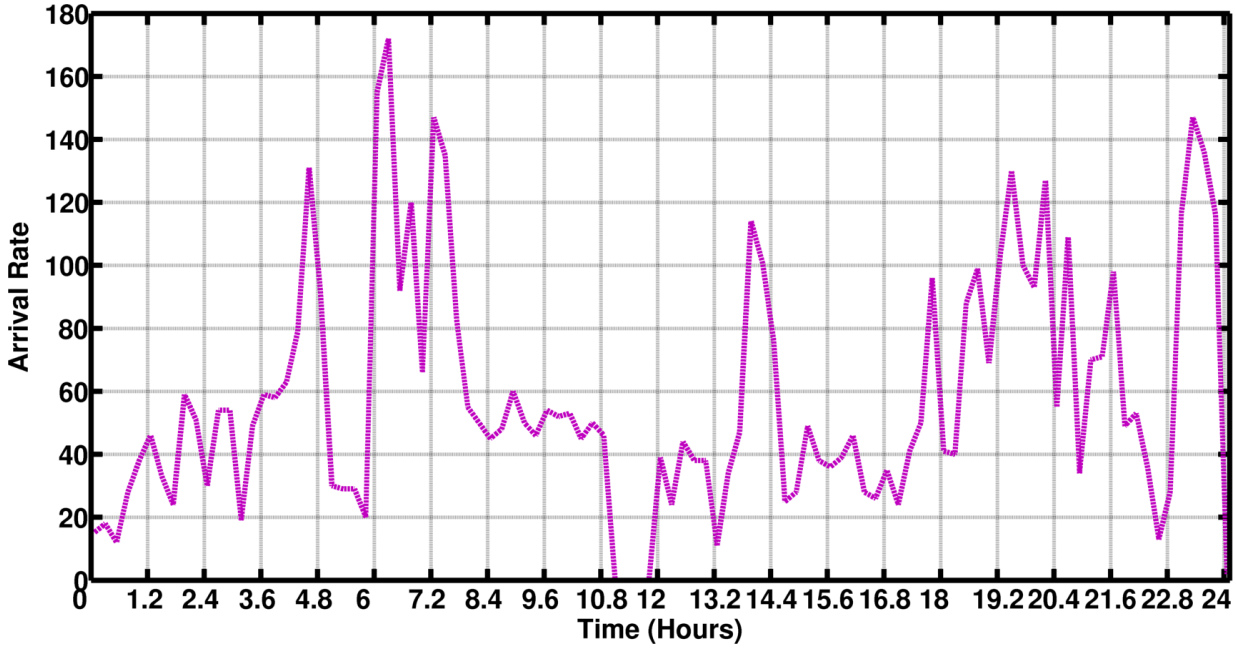


Figure 2.1: Graph showing the variation in arrival rate over a 24 hour period.

Many companies provide separate cloud computing units and dedicated data-centers [8] to create cloud computing solution for specific industries [9]. This would imply that dedicated servers would run similar tasks. In this thesis, we focus on a similar scenario in which similar types of jobs are to be processed by a dedicated server.

2.2 The Dark Silicon Era

Computing has reaped exponential increases in performance in the past few decades due to Moore's law [10](the doubling of transistors on a chip every 18 months) combined with Dennard scaling [11](power density remains constant as transistor size reduces) along with advances in fields like micro-architecture and compilers. Around 2005 processor designers moved away from concentrating on single-core performance to multi-core in an effort to keep up with the increase in the number transistor and improve performance accordingly.

In the multi-core era, researchers have looked to build systems with hundreds of core. But only a few applications are able to exploit this level of parallelism. This predicament has gone on to pose problems in parallel programming and architecture design at a big scale.

In addition, the failure of Dennard scaling has brought about a barrier which Venkatesh et al. [12] refer to as the utilization wall: with each successive process generation, the percentage of a chip that can switch at full frequency drops exponentially due to power constraints. The percentage of the chip that cannot be turned on is termed "dark silicon". It is widely expected that the area of the chip that is be "dark" will be more than half of the total area of the chip in the near future [13].

As most of the chip will not be active in the near future, researchers have shifted on to finding ways to better use the passive area available to them. The two most widely used techniques to alleviate the problem of dark silicon.

- **Dim Silicon.** The idea here is to under-clock general purpose hardware to meet the power constraints [14]. Hence, the term "dim" silicon.
- **Specialized Cores.** The idea here is to dedicate specialized hardware like accelerators and co-processors for improved power efficiency of certain programs or parts of programs [12].
- **Heterogeneity.** The idea here is to provision the passive area available on the chip with different types of cores (all the cores cannot be turned on at the same time due to power constraints) and decide which cores to turn on based on power budget and need [15].

2.3 DoP optimization

Prior work has looked at optimizing the DoP in the context of minimizing the execution time of parallel database queries and streaming applications. In the case of parallel database queries, Rahm [16] made the observation that the optimal degree of scan parallelism depends on arrival rate. When considering streaming applications, Raman et al. [17] propose DoPE, an Application Programming Interface (API) and run time system, that allows the application developer to develop a functionally correct parallel program without having to worry about optimizing the parallelism for different environments. They also make the observation that the optimal degree of parallelism is dependent on load on the system. These papers do not formalize these notions using queueing theory.

Li and Martinez [18] looked at the problem of optimizing power consumption of parallel applications under certain constraints by changing the DoP and applying dynamic voltage/frequency scaling. Sasaki et al. [19] suggest a scalability based many-core partitioning

scheduler which assigns the optimal number of cores depending on the scalability of the application. These papers do not model job arrivals and therefore ignore the queueing delay component of service time. Furthermore, all the approaches highlighted above are in the context of homogeneous multi-cores and do not apply to asymmetric multi-cores.

2.4 Motivation for Heterogeneous Multi-core Processors

Kumar at el. [20] were one of the first to propose the idea of heterogeneous multi-core processors and performed thread scheduling by matching the characteristics of threads to cores. It is assumed that each core has the same instruction set architecture (ISA) but different micro-architectural parameters. However, they do not optimize for DoP and do not model job arrivals and queueing delays. Turakhia et al. [15] have addressed the problem of optimally synthesizing heterogenous multi-core processors to maximize performance within the given area and power budgets for a range of applications. Although the authors do optimize the DoP of each application, job arrival rates are not modelled.

There has also been recent work looking explicitly at the type of clustered asymmetric multi-core processors that we focus on in this thesis, but these papers exploit phase behaviour in application characteristics *within* an application. Ahn at el. [21] presented McSimA+, a high-level simulator for asymmetric clustered many-core architecture. As a case study, the authors study a mechanism to predict which cluster will provide the best performance depending on the execution phases and migrate accordingly [22]. Muthukaruppan et al. [23] propose a power management framework for asymmetric multicore architectures which aims to provide satisfactory user experience but at the same time maintain energy consumption within the thermal design power (TDP) constraint.

An example of an asymmetric clustered processor which is currently available is the Exynos processor, based on ARM's big.LITTLE architecture. It is used in a variety of handheld devices. The processor consists of two clusters, one which is made of four high performance Cortex A-15 cores and the other is made of four power-efficient Cortex A-7 cores. In the simplest mode of operation only one cluster can be turned on at any point in time. Here heterogeneity is used to provide power efficiency and high performance when needed. Overview of the processor is shown in Figure 2.2.

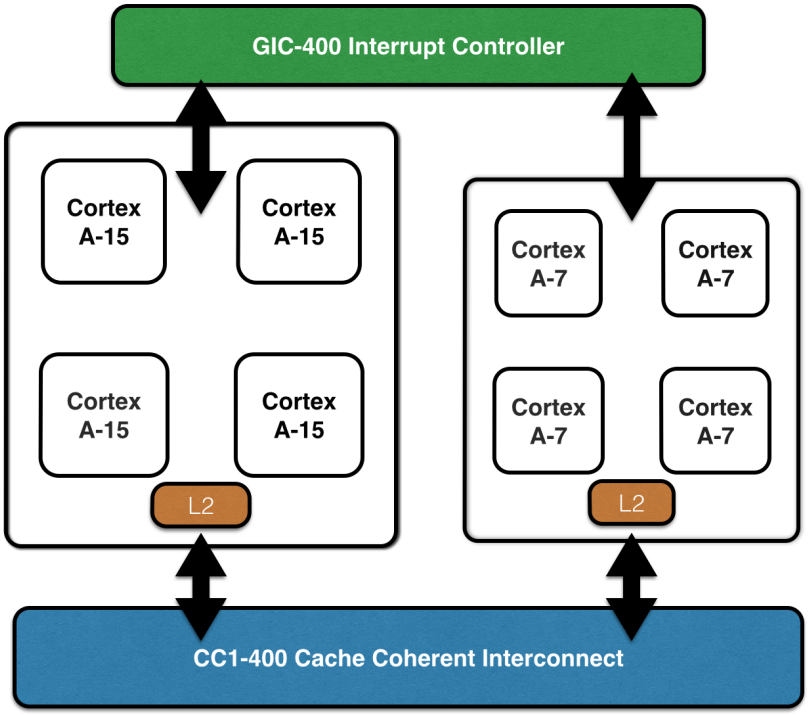


Figure 2.2: Overview of the Exynos processor as seen in [4]

In general, the motivation for heterogeneous architectures in the prior work has been diversity in application characteristics. This thesis builds a case for heterogeneity based solely on the variations in job arrival rate, i.e., even if a processor always executes the same type of jobs.

Chapter 3

Approach

In this chapter, we give a brief introduction to queueing theory, propose a queueing theoretic model for mean service time, and a job arrival rate aware runtime scheduler.

3.1 Introduction to Queueing theory

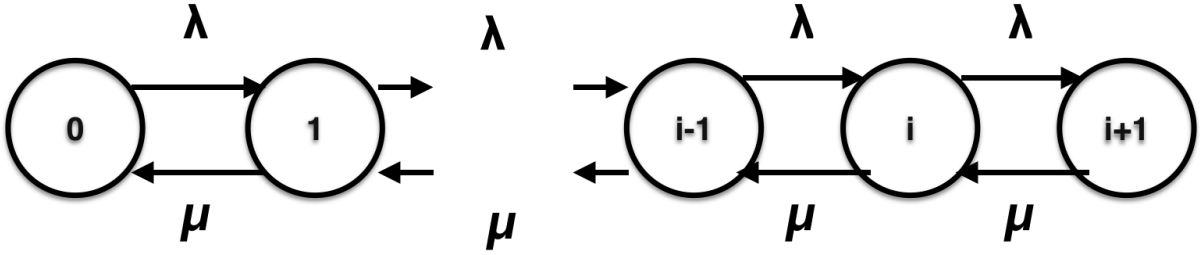


Figure 3.1: A diagram explaining the states in queueing theory

Queueing theory, the study of queues, provides us with a way of modelling different scenarios. For instance, consider customers waiting for a service in a bank. Here customers

come in to the queue, wait their turn to get serviced and leave the queue once serviced. The rate at which customers enter the queue is referred to as the arrival rate, denoted by λ and the customers are served on a first come first serve basis. The rate at which the bank teller, also called servers, is able to service the customers is referred to as the service rate, denoted by μ. In this particular scenario there might be a few variations,

- There might be more the one servers who are able to service the customers
- There might be more than one queue the customer forms into
- There might be a combination of the above
- There might be a restriction on the number of people in the queue

The simplest model possible is an M/M/1 model. Here there is a single server and the customer/jobs arrive according to a Poisson process. The service times of the jobs are exponentially distributed. This model can be extended to an M/M/n where n queue stands for the number of servers. For our purpose we use a M/M/n queue as there can be multiple servers as we will see shortly. In our case we also assume that the queue length is infinite.

Figure 3.1 shows the different states in a simple queueing system (also known as a state space diagram). It shows a Markov chain, with each number representing the number of customers/jobs in the line. This process is a special case of the Markov chain know as birth-death process as only two types of transitions are possible.

3.1.1 Important Terms

The total service time. The total service time, denoted as W, is defined as the sum of the time spent waiting in the queue, W_q, and service time, S, when the job gets assigned to a server:

$$W = W_q + S. \tag{3.1}$$

Server Utilization. The server utilization, ρ, for a given arrival rate, λ, is defined as

$$\rho = \frac{\lambda}{n * \mu} \tag{3.2}$$

For the queue to be in a stable state ρ should be less than one.

Maximum sustainable arrival rate. To avoid the queue length from blowing up each system has a maximum sustainable arrival rate λ_{max} this is defined as the product of the number of servers and the service rate

$$\lambda_{max} = n * \mu. \tag{3.3}$$

3.2 Queueing Theoretic Modelling

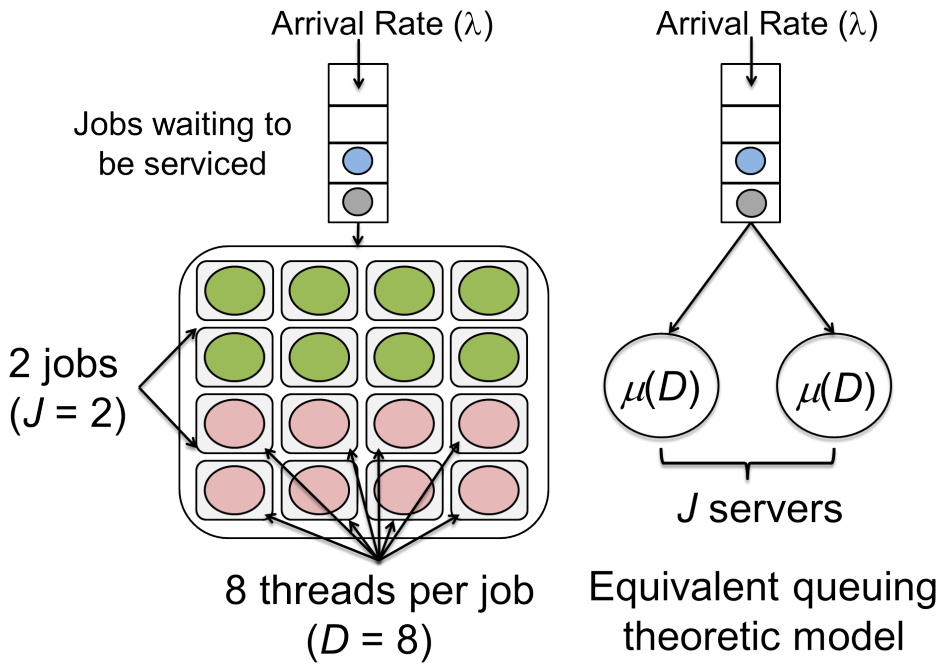


Figure 3.2: A multi-core processor with $N_c = 16$ cores. The scheduler schedules two jobs ($J = 2$) on the processor, each with a DoP of three ($D = 8$). Also shown on the right is an equivalent queueing theoretic model of the processor.

When considering the cloud setting, one in which jobs are constantly arriving to be serviced, an important metric is the total service time, i.e., the sum of the time spent waiting in the queue for service and the execution time on the processor. We propose a

queueing theoretic analytical model (validated against detailed simulations in the chapter on experimental results - chapter 5) for the mean total service time on a multi-core processor as a function of job arrival rate and the DoP of each job.

We look at a single cluster in the asymmetric processor consisting of N_c homogeneous cores processing jobs that arrive at the server for processing at an arrival rate λ, which follows a Poison process. Each job is assigned a DoP of D and up to $J = \frac{N_c}{D}$ jobs can be processed in parallel. For example, In a cluster containing 32 cores, one could have a DoP (D) of 4 and 8 jobs (J) running in parallel or have a DoP of 16 and 2 jobs running in parallel.

We define $\mu(D)$ to be the mean service rate (inverse of execution time) when a job is processed with a DoP of D, and assume that the job size is exponentially distributed. This can be modelled as an M/M/n queue (a queueing system with n servers) with number of servers $n = J$ as shown in Figure 3.2.

Under these assumptions, we can use standard results from queueing theory to write the mean total service time, W, of jobs as follows [24]:

$$W = \frac{p\left(J, \frac{\lambda}{\mu(D)}\right)}{J\mu(D) - \lambda} + \frac{1}{\mu(D)} \tag{3.4}$$

where $p\left(J, \frac{\lambda}{\mu(D)}\right)$ is the probability that a job has to wait in the queue to be serviced. To determine $p\left(J, \frac{\lambda}{\mu(D)}\right)$, we first rewrite the utilization of the server, ρ, as:

$$\rho = \frac{\lambda}{J \times \mu(D)}. \tag{3.5}$$

We can now write $p\left(J, \frac{\lambda}{\mu(D)}\right)$ as:

$$p\left(J, \frac{\lambda}{\mu(D)}\right) = \frac{\frac{(J\times\rho)^J}{J!} \times \frac{1}{1-\rho}}{\sum_{K=0}^{J-1} \frac{(J\times\rho)^K}{K!} + \frac{(J\times\rho)^J}{J!} \times \frac{1}{1-\rho}}. \tag{3.6}$$

3.2.1 Performance modelling for multi-threaded applications

We focus on multi-threaded applications. These applications consist of two phases of execution — a sequential phase, which consists of a single thread of execution; and a parallel phase in which multiple threads process data in parallel. The parallel threads of execution in a parallel phase typically synchronize on a barrier, in other words, all threads must finish execution before the application can proceed to the next phase. Therefore, the latency of a parallel phase is dominated by the worst case execution latency across all parallel threads. Based on this observation, we model the execution time of an application, E, as

$$E = \frac{W_{seq}}{f_{seq}} + \frac{W_{par}}{D \times \min_{i \in [1,D]}(f_{par,i})}, \tag{3.7}$$

where W_{seq} and W_{par} represent the sequential and parallel components of the application, respectively. D is the number of parallel threads in the application. Furthermore, f_{seq} and $f_{par,i}$ refer to the frequency at which the sequential and the i^{th} parallel thread are executed, respectively. These values depend on the scheduling of threads to cores in the multi-core processor, *and* the frequency assigned to each core. It should be noted that W_{seq} and W_{par} are computed based on each core type. It can be seen that this equation is based on Amdahl's law [25].

When considering a set of homogeneous cores, Equation 3.7 can be rewritten to represent $\mu(D)$ as

$$\mu(D) = \frac{\mu_{seq}}{S + \frac{1-S}{D}} \tag{3.8}$$

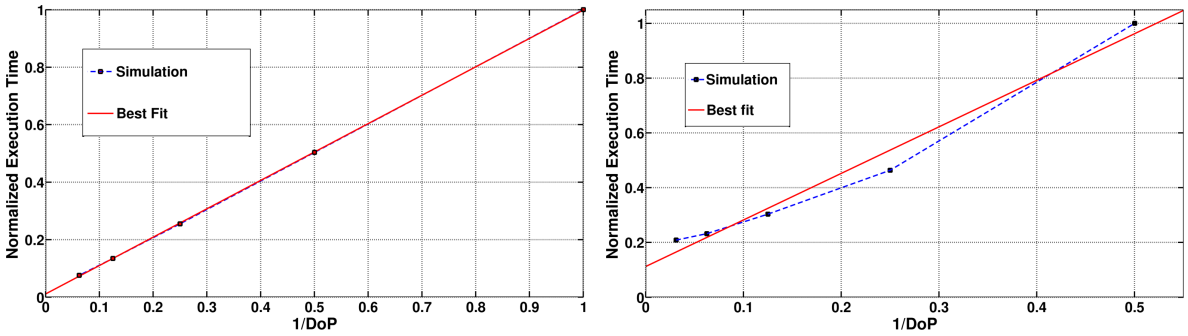
where μ_{seq} is the service rate if the job is executed sequentially, and S is the fraction of execution that cannot be parallelized. S is also referred to as the *serial fraction*. Although the model is simple, it is in fact very accurate for a variety of multi-threaded benchmarks as illustrated in Figure 3.3

μ_{seq} and S in terms from 3.7 are

$$\mu_{seq} = \frac{f}{W_{seq} + W_{par}} \tag{3.9}$$

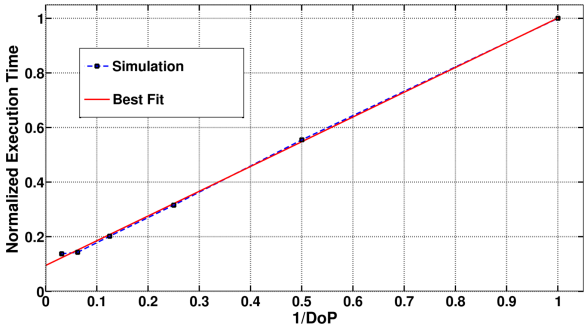
$$S = \frac{W_{seq}}{W_{seq} + W_{par}}. \tag{3.10}$$

As we are considering homogenous cores, there is no need to distinguish between f_{seq} and f_{par} as in Equation 3.7. Hence Equation 3.9 has a single term f denoting frequency of the cores.

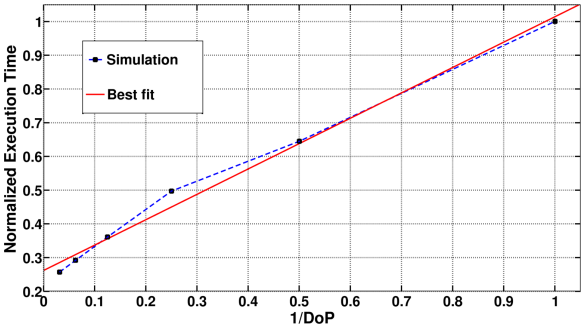


(a) Radix

(b) Blackscholes



(c) LU.ncont



(d) PCA

Figure 3.3: Normalized execution time vs degree of parallelism for the (a) Radix (b) Blackscholes (c) LU.ncont (d) PCA benchmarks from cycle-accurate simulations and best fit based on Equation 3.7.

3.2.2 Analysis

Based on this queueing theoretic model, we will begin by analyzing the optimal DoP for perfectly parallelizable jobs, i.e., when $S = 0$.

Remark 1. *For $S = 0$, the optimal DoP, $D^* = N_c$, and $J^* = 1$ for any job arrival rate.*

Proof. When $S = 0$, $\mu(D) = D\mu_{seq}$, i.e., the service rate increases linearly with D. A standard result from queueing theory [24] shows that n servers with service rate μ have larger mean service time than a single large server with service rate $n\mu$, for any choice of n. Thus as D increases, the mean waiting time decreases. Since the maximum value of D is the number of cores N_c, $D^* = N_c$. □

Remark 1 effectively says that if the jobs can be perfectly parallelized, the optimal choice is *always* to execute jobs in sequence, and to execute each job with the maximum amount of parallelism, independent of arrival rate. Figure 3.4a empirically validates this result.

We will now consider what happens in practical scenarios where the serial fraction is non-zero, i.e., $S > 0$. We analyze the low arrival rate and the high arrival rate regimes separately.

Remark 2. *For $S > 0$ and $\lambda \to 0$, the optimal DoP, $D^* = N_c$ and $J^* = 1$.*

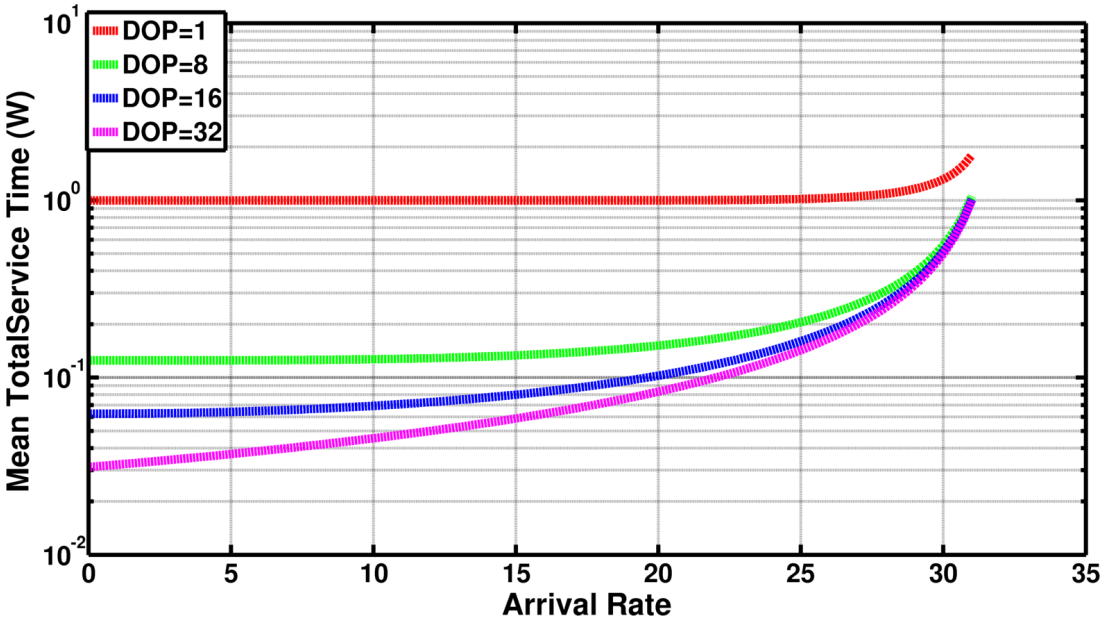
Proof. As $\lambda \to 0$, $W \to \frac{1}{\mu(D)}$; thus the service time is minimized when $\mu(D)$ is maximum, i.e., when $D^* = N_c$. □

Remark 3. *For $S > 0$, the optimal DoP that sustains the highest arrival rate is $D^* = 1$, and $J^* = N_c$.*

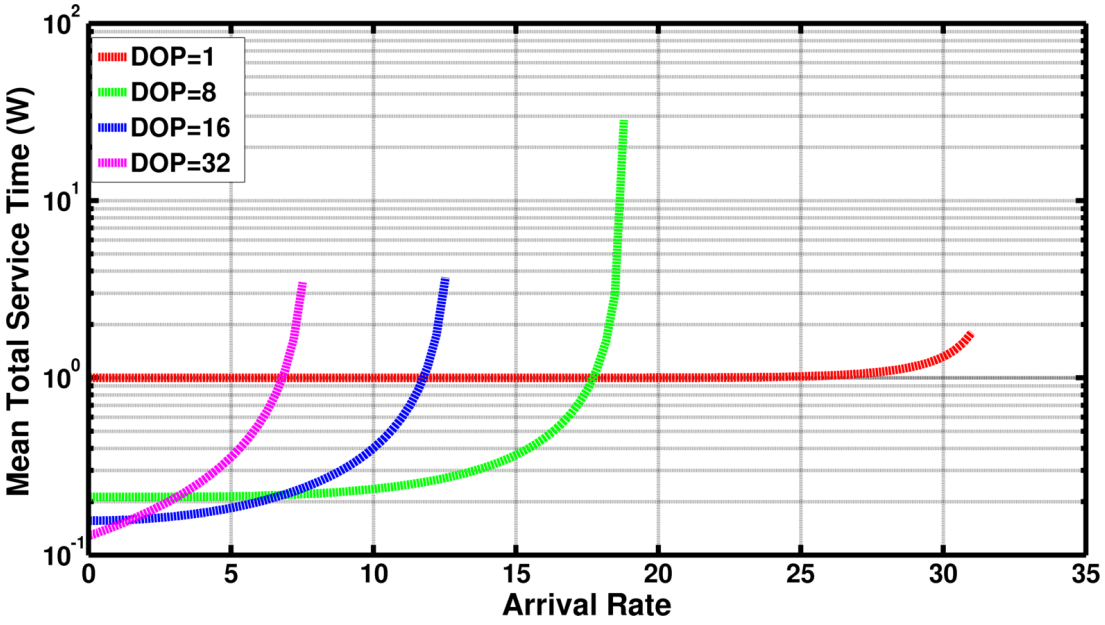
Proof. The maximum sustainable arrival rate for any choice of D is:

$$\lambda_{max} = J\mu(D) = \frac{N_c}{D}\mu(D) = \frac{N_c\mu_{seq}}{DS + (1 - S)}$$

and therefore, λ_{max} reduces as D increases for $S > 0$. Thus the optimal DoP for high arrival rates is $D^* = 1$. □



(a) $S = 0$



(b) $S = 0.1$

Figure 3.4: Mean total service time as a function of arrival rate for different DoPs: (a) perfectly parallelizable jobs; (b) jobs with 10% serial fraction. This is for a system with 32 cores.

Figure 3.4b shows the mean total service time versus arrival rate for different DoPs with $S = 0.1$. It can be observed that, in fact, the optimal DoP reduces with increasing job arrival rates.

Cluster Migrations with Varying Arrival Rate. So far, we have discussed the optimal DoP as a function of arrival rate given N_c homogeneous cores. More generally, assume that there exist T types of cores on the chip, and correspondingly T clusters.

Assume that the peak power consumption of a core $t \in T$ is given by P_c^t, and that the chip wide power budget (excluding uncore components) is given by P_{budget}. Thus, the number of cores of type t that can be allocated in a cluster and turned on simultaneously is:

$$N_c^t = \lfloor \frac{P_{budget}}{P_c^t} \rfloor$$

Finally, assume that the mean sequential execution time of a job on a core of type t is μ_{seq}^t.

As before, we start by examining the scenario when $\lambda \to 0$. We have already shown that the optimal DoP for this case is the maximum DoP. We can therefore write the service rate as:

$$\mu\left(D = N_c^t\right) = P_{budget} \frac{\mu_{seq}^t}{SP_{budget} + (1-S)P_c^t}$$

In this equation, since P_{budget} is a constant, the optimal choice of core is the core that maximizes:

$$\frac{\mu_{sec}^t}{SP_{budget} + (1-S)P_c^t}$$

Observe the choice for optimal core depends on the serial fraction S and is not, in general, the most power efficient core. As S increases, the optimal core choice is biased towards larger, faster cores. This is because the larger, faster cores tend to have higher execution when executed sequential, i.e. the term μ_{seq}^t and the term $(1-S)P_c^t$ becomes less significant.

Conversely, for high arrival rates, we have seen that the optimal DoP choice is $D = 1$. The maximum arrival rate that a core $t \in T$ can support is given by:

$$\lambda_{max}^t = P_{budget} \frac{\mu_{seq}^t}{P_c^t}$$

It is clear from the above equation that the optimal choice for high arrival rates is the most power efficient core, i.e., the one that maximizes $\frac{\mu_{seq}^t}{P_c^t}$.

These observations motivate the need for cluster migrations as the job arrival rate changes; as we will see, as the arrival rate increases, jobs are migrated from clusters with fewer higher performance, higher power cores to clusters with a larger number of low performance, low power cores.

3.2.3 Impact of multiple jobs running in parallel

When the DoP is quite low, more jobs are able to run in parallel. As more jobs run in parallel the average execution time per job increases, this is due to increased contention for shared resources. This is shown in Figure 3.5. Note that the increase in average execution time is *steeper* for lower DoPs than for higher DoPs.

In the specific case for the FFT benchmark shown in Figure 3.5, going from D=4 to D=1 with only one job running on a pod with 32 cores results in 41% increased service time. However, when all 32 cores are occupied, i.e., 32 parallel jobs in the $D = 1$ case and 8 parallel jobs in the $D = 4$ case, the service time increases by 68%. Thus although theoretically, $D = 1$ is optimal for high arrival rates, in practice we observe that slightly higher DoPs are preferred in many cases.

Typically, lower DoPs are optimal for higher arrival rates due to fact that they allow more jobs to run in parallel. On the other hand higher DoPs are optimal for lower arrival rates when there is not a need to run more jobs in parallel. In either case the number of cores occupied is high. Thus, when we consider average service time for a particular DoP it is the average service time when all cores are occupied. For example, in a 32 core machine for a DoP of 32 all cores are occupied when only one job is running. For a DoP of 4, all cores are occupied when 8 jobs are running in parallel.

To further illustrate this point, we perform an analytical experiment where we track the percentage of cores occupied over a range of arrival rates. For this experiment we choose a job with a serial fraction (S) of 0.1 and this runs on a system with 32 cores. Figure 3.6 shows the results. We define occupancy as the average percentage of cores occupied when at least one job is running. The figure also shows the change in optimal DoP as the arrival rate changes. We observe that across arrival rates, the occupancy is always greater than 50%.

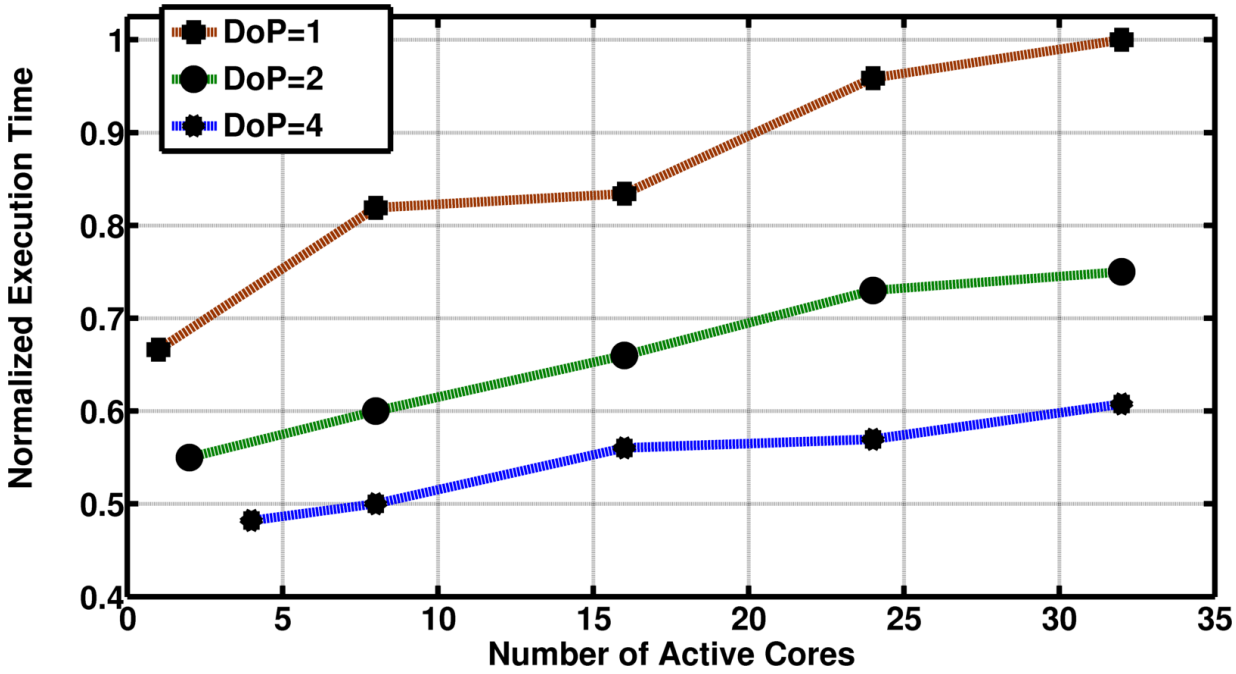


Figure 3.5: Increase in execution time with increase in number of jobs running in parallel. The values shown above are for FFT benchmark

3.3 Job Arrival Rate Aware Scheduler

We now describe the proposed job arrival rate aware run-time scheduler. The run-time scheduler monitors the job arrival rate and based on the observed arrival rate, makes two decisions: (i) which cluster to migrate to, and (ii) the optimal DoP (D) and number of parallel jobs (J) for that cluster.

We assume that the run-time scheduler has pre-characterized information about the service rate for each cluster type and DoP combination; i.e., $\mu^t(D, B)$ is known. The scheduler then uses the Equation 3.4 with $n = \frac{N_c^t}{D}$ to determine the average service time (waiting time plus execution time) and picks the cluster and DoP combination with the lowest mean service time for the given job arrival rate. The time complexity of evaluating this expression is linear in the number of cluster types and the maximum number of cores in any cluster.

There are two practical challenges in implementing the run-time scheduler that still need to be addressed. First, the job arrival rate for any given scheduling interval is not

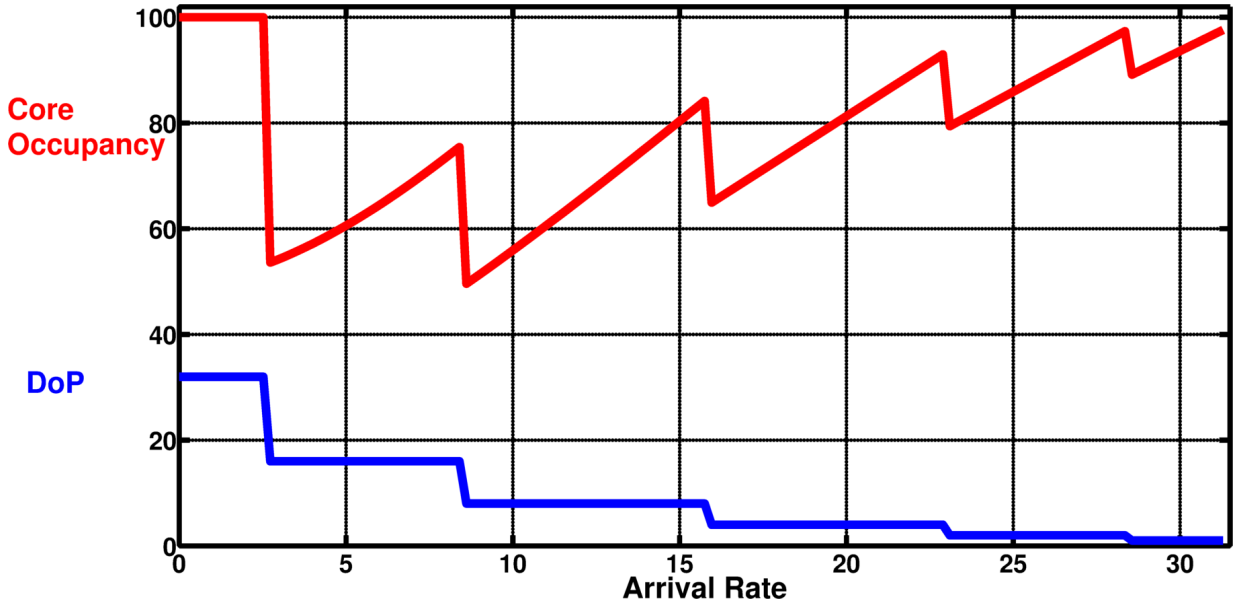


Figure 3.6: Graph showing the percentage of cores occupied for the optimal DoP for a range of arrival rates. Serial fraction the job(S) is 0.1

known in advance, so we need a mechanism to predict the arrival rate. Second, if a cluster migration decision is made, the overheads of migration need to be accounted for.

Arrival Rate Prediction. We use a naive predictor, i.e., the arrival rate in the current scheduling interval is predicted to be the same as the arrival rate in the previous control interval. More sophisticated prediction mechanisms can be designed, but that is demarcated as future work. In addition, as we note in our experimental results, even the simple prediction mechanism provides competitive results compared to a scheduler with oracular knowledge of arrival rates.

Cluster Migration Mechanism. When the scheduler decides to migrate jobs to a new cluster, we first wait till all jobs currently executing on the current cluster finish executing. When there are no jobs remaining on the current cluster, the parent kernel thread is migrated to the new cluster.[1] Finally, the jobs in the scheduler queue are executed with the optimal DoP for the new cluster.

Cluster migration incurs an additional performance overhead because (i) jobs are held up in the queue as currently executing jobs finish execution; (ii) the overhead of migrating

[1]Asymmetric multi-core processors such as the ARM big.LITTLE are architected with a special hardware block called the Generic Interrupt Controller (GIC) to migrate the parent thread between clusters [4].

the parent thread state (estimated to be $50\mu s$ by [4]); and (iii) the overhead of starting with cold caches in the new cluster. To ensure that the overheads do not negate the benefits of cluster migration, we only migrate when the predicted service time benefits from migration are more than a statically determine threshold, set to 10% in our experimental results. We have accounted for all sources of overhead in our empirical evaluation, as discussed in the experimental results chapter (chapter 5) .

Chapter 4

Experimental Setup

We experimentally evaluate a candidate asymmetric multi-core architecture consisting of three clusters and a 16 MB LLC at the 11 nm technology node, where more than half the chip is expected to be dark [3]. The cores in each of the three clusters are based on the Intel Nehalem micro-architecture. Table 4.1 provides the micro-architectural parameters of the cores in each of the three clusters.

Power and area numbers for the cores are obtained from the McPAT tool [26] for the 22 nm process node and scaled to 11 nm process node using the scaling factors indicated by [3]. We conservatively estimate that at 11 nm, each MB of LLC dissipates 0.5 W of power (more details in Table 4.2. Since we are interested in server class processors, the total chip power budget is set to 115 W, of which 85 W is for the core components and the remaining for the uncore components. At the given power budget, the small (S), medium (M) and large (L) clusters accommodate 64, 32 and 16 cores, respectively.

To obtain experimental results, we make use of the *Sniper* multi-core simulator [27] and test a wide range of applications from the SPLASH-2 benchmark suite [28], the PARSEC

Core Type	Nominal Frequency	Dispatch Width	Window Size	Peak Power Consumption	L1-D Size	L1-I Size	Normalized Area
Small	4.5 GHz	1	16	$1.29W$	64 KB	64 KB	1×
Medium	4.5 GHz	2	128	$2.59W$	64 KB	64 KB	1.33×
Large	4.5 GHz	4	128	$5.18W$	128 KB	128 KB	1.66×

Table 4.1: Core micro-architectural details.

L3 Cache Sizes	Area	Power
4 MB	2.6 mm^2	$2W$
8 MB	4.2 mm^2	$4W$
16 MB	8.2 mm^2	$8W$

Table 4.2: Last level cache details.

Application	Description
PARSEC Benchmark Suite	
Blackscholes	Financial option pricing
Bodytrack	Image processing
Phoenix Benchmark Suite	
K-means	Clustering
Linear regression	Statistical machine learning
PCA	Statistical machine learning
String match	Text processing
Word count	Text processing
SPLASH-2 Benchmark Suite	
Barnes	Simulation of interaction
Cholesky	Sparse matrix factorization
FFT	Fast Fourier transform
LU.ncont	Dense matrix factorization
Radix	Sorting
Raytrace	Image rendering

Table 4.3: Application Details

benchmark suite [29] and the Phoenix benchmark suite [30] (more details in Table 4.3). The applications selected are representative of the workloads that we expect to see in the cloud setting and range from the finance to machine learning, text processing, and image processing applications. The Phoenix benchmark suite, that is based on map-reduce, is particularly representative of cloud computing workloads, although applications from the SPLASH-2 and PARSEC benchmark suite are quite frequently used in the cloud setting [31].

To determine the mean total service time, we need to simulate thousands of jobs as they arrive for processing, which is prohibitively time-consuming to do in a detailed micro-architectural simulator like *Sniper*. Instead, we use a hybrid approach that combines execution time data from the *Sniper* with a Python based discrete event simulation (DES) engine based on SimPy.[1] We call this more practical approach *Sniper*+DES. A similar simulation approach was proposed by [32].

[1] http://simpy.sourceforge.net/

Chapter 5

Results and Evaluation

5.1 Model Validation

Validating *Sniper*+DES We validate the *Sniper*+DES approach against a golden baseline in which we set up an end-to-end simulation using *Sniper* alone to simulate hundreds of jobs as they arrive for execution at a given arrival rate.[1] Figure 5.1 compares the mean service time versus arrival rate for *Sniper* and *Sniper*+DES. We observe that the mean total service time obtained from *Sniper*+DES is within 8.25% of *Sniper* over a wide range of arrival rates, while being *several* orders of magnitude faster in terms of simulation time. The rest of the experimental data is presented using *Sniper*+DES simulations.

Validating Analytical Models Figure 5.1 also shows the mean total service time versus arrival rate obtained from the proposed analytical models versus simulations. As it can be seen, the two are in excellent agreement over a wide range of arrival rates. Recall that the run-time scheduler uses the analytical models to make scheduling decisions, which is why the accuracy of these models is particularly relevant.

[1] A single end-to-end *Sniper* simulation can take days for the larger benchmarks.

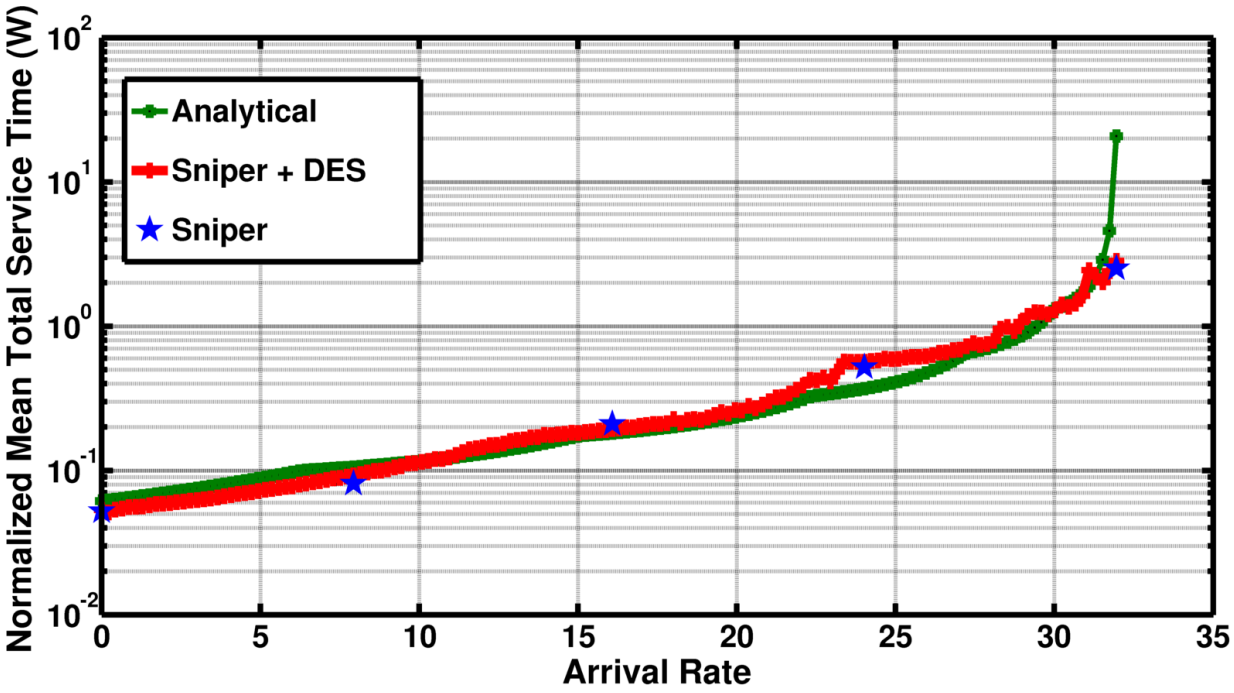


Figure 5.1: Mean total service time predictions from analytical model, DES + Sniper and the Sniper multi-core simulator for the Radix benchmark - Medium Core.

5.2 Job Arrival Rate Aware Scheduling Results

We begin by validating empirically our observations about the arrival rate dependence of optimal DoP and optimal cluster.

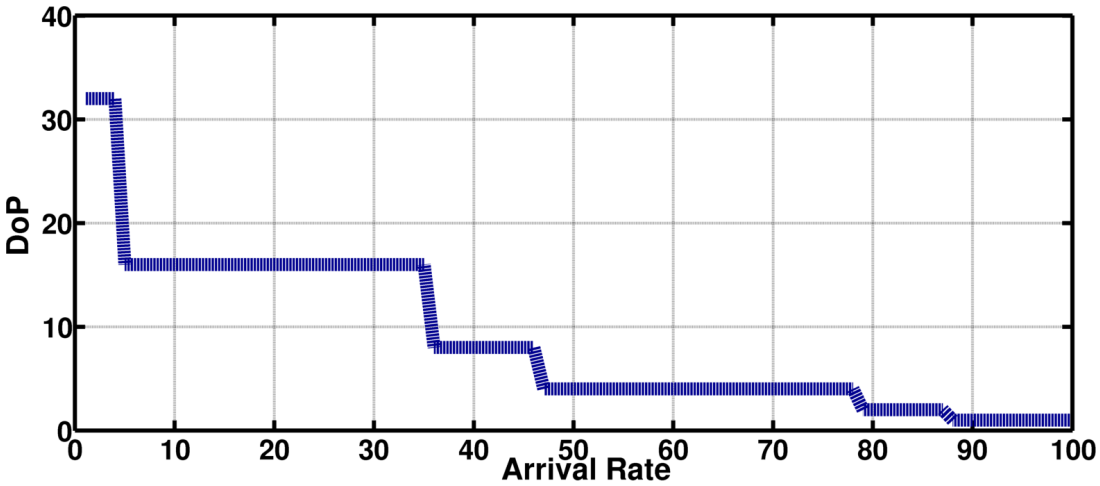
Optimal DoP Decreases With Arrival Rate Figure 5.2a shows the optimal DoP configuration for the FFT benchmark executing on a cluster with 32 medium cores, across a range of arrival rates. Observe that, as predicted, the optimal DoP reduces with increasing arrival rate. A similar plot is seen in Figure 5.2b for the Raytrace benchmark executing on a cluster with 16 big cores.

Optimal Cluster Depends on Arrival Rate In Figure 5.3a we show the service time versus arrival rate curves on all three clusters for the Radix benchmark. It can be observed that for low arrival rates, the cluster with medium sized cores is used first, and as the arrival rate increases, the cluster with small sized cores is used. In the other example in Figure 5.3b, it can be observed that for low arrival rates, the cluster with large cores is used first, and as the arrival rate increases, the cluster with medium sized cores is used. This validates the observation that as the arrival rate increases, clusters with smaller, more power efficient cores become optimal.

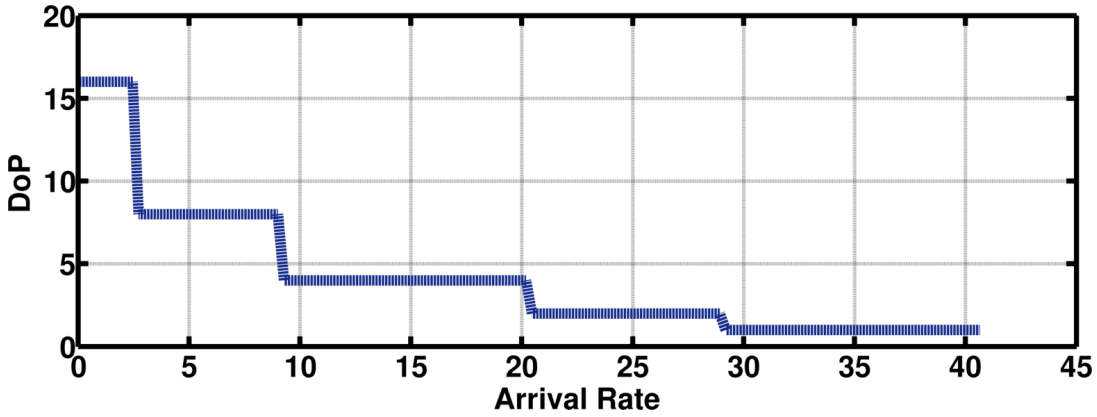
Results with Real Arrival Rate Data To account for job arrival rate variations, we use real world data obtained from a Facebook map-reduce cluster that traces the arrival rate variations in a 24 hour period [33], as shown before in Figure 2.1. For each benchmark the arrival rate curve is scaled to its maximum sustainable arrival rate.

With this arrival rate curve as input, we experimented with three different schedulers:

- A naive, baseline scheduler that determines for each benchmark the best static DoP and best static cluster to execute on, i.e., the DoP and cluster do not change as the arrival rate varies.
- A scheduler that statically determines the best cluster for each benchmark, *but* varies the DoP based on job arrival rate. We show data when both oracular knowledge of arrival rate is available, and when the arrival rate is predicted using the proposed prediction mechanism.

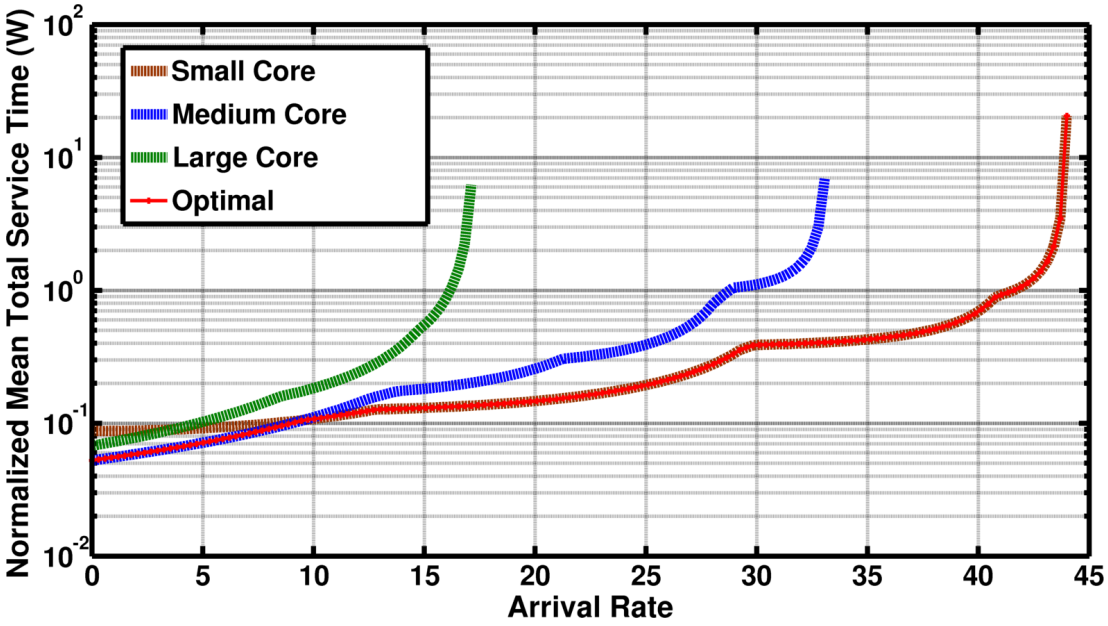


(a) FFT

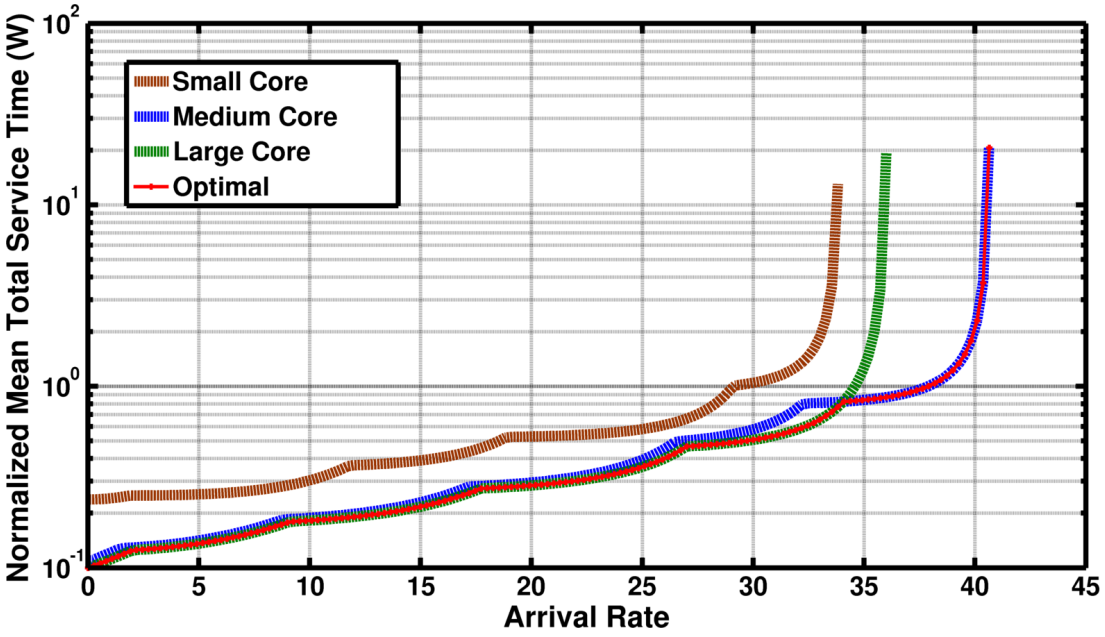


(b) Raytrace

Figure 5.2: Optimal DoP as a function of arrival rate for the (a) FFT (b) Raytrace benchmarks.



(a) Radix



(b) Raytrace

Figure 5.3: Mean total service time versus arrival rate for three different clusters for (a) Radix (b) Raytrace benchmarks.

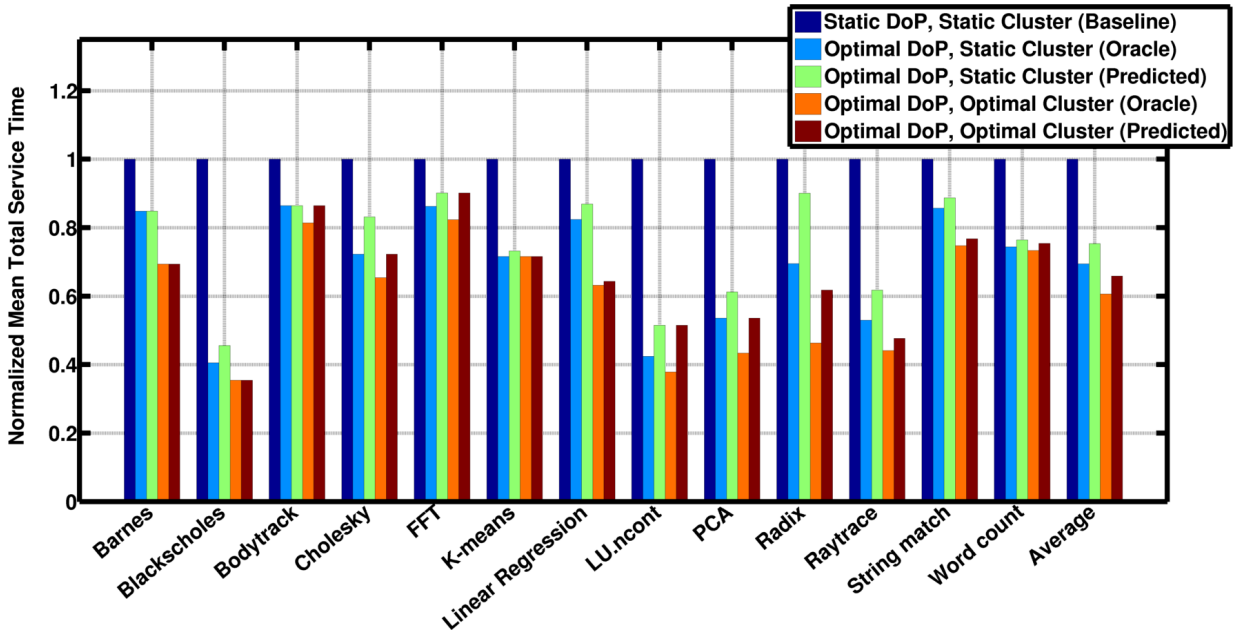


Figure 5.4: The percentage improvement in the mean total service time over all benchmarks with oracular knowledge of the arrival rate and using the naive prediction of the arrival rate with baseline as the single best cluster using the single best DoP

- A scheduler that varies both the DoP and cluster type (i.e., performs cluster migrations) based on job arrival rate. We show data when both oracular knowledge of arrival rate is available, and when the arrival rate is predicted using the proposed prediction mechanism.

Figure 5.4 plots the mean total service time for different benchmarks for different scheduling approaches, normalized to the baseline scheduler. We note that averaged over benchmarks, 25% improvement is observed by optimally selecting the DoP based on arrival rate, and 34% improvement is observed using both optimal DoP selection and cluster migration. If oracular arrival rate information were available, the improvements would be 31% and 39%, respectively. The naive predictor was only 8.7% worse off than with the oracular knowledge when optimally selecting the DoP and only 8.1% worse off than the oracular knowledge when optimally selecting both DOP and cluster migration.

Finally, we note that although we have so far discussed only improvements in *mean* service time, service level agreements (SLA) for performance are sometimes based on X^{th} percentile service time [31] (X is typically 95%).

In the context of X^{th} percentile SLAs, we note that the proposed job arrival rate aware scheduler outperforms the baseline scheduler on this metric as well. This can be seen in Figure 5.5 which shows the CDF of the service time of the Radix benchmark under the real world arrival rate curve for the optimal, job arrival rate aware scheduler versus schedulers that statically select the small, medium and large clusters. It can be seen that the proposed optimal scheduler outperforms static schedulers for any percentile SLA. Figure 5.6 gives the CDF of the service time of the Barnes benchmark, which shows the same trend.

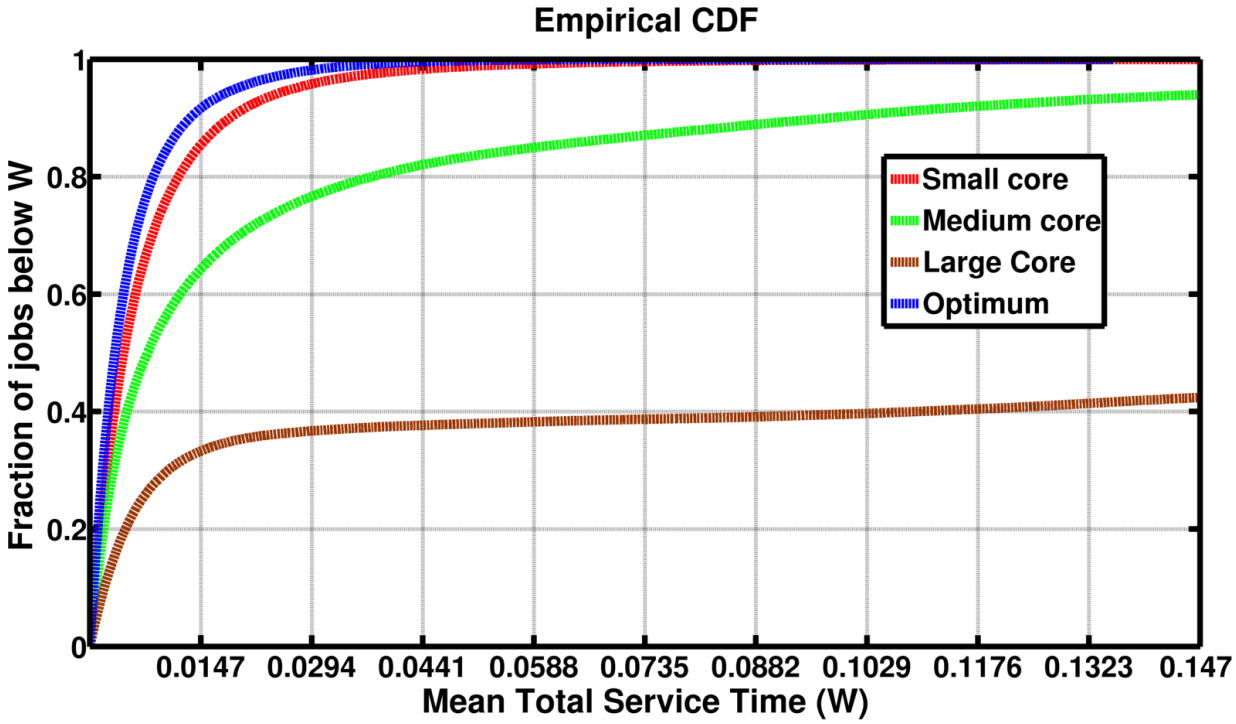


Figure 5.5: Cumulative distribution function of total time spent in the system for the Radix benchmark

Overhead of Cluster Migration. To understand how frequently the run-time scheduler updates the DoP and/or migrates to a new cluster, we plot in Figure 5.7a the optimal DoP and optimal cluster type as a function of time for the Radix benchmark. Note that for this particular benchmark, the large core type is not used. However, it is used for the

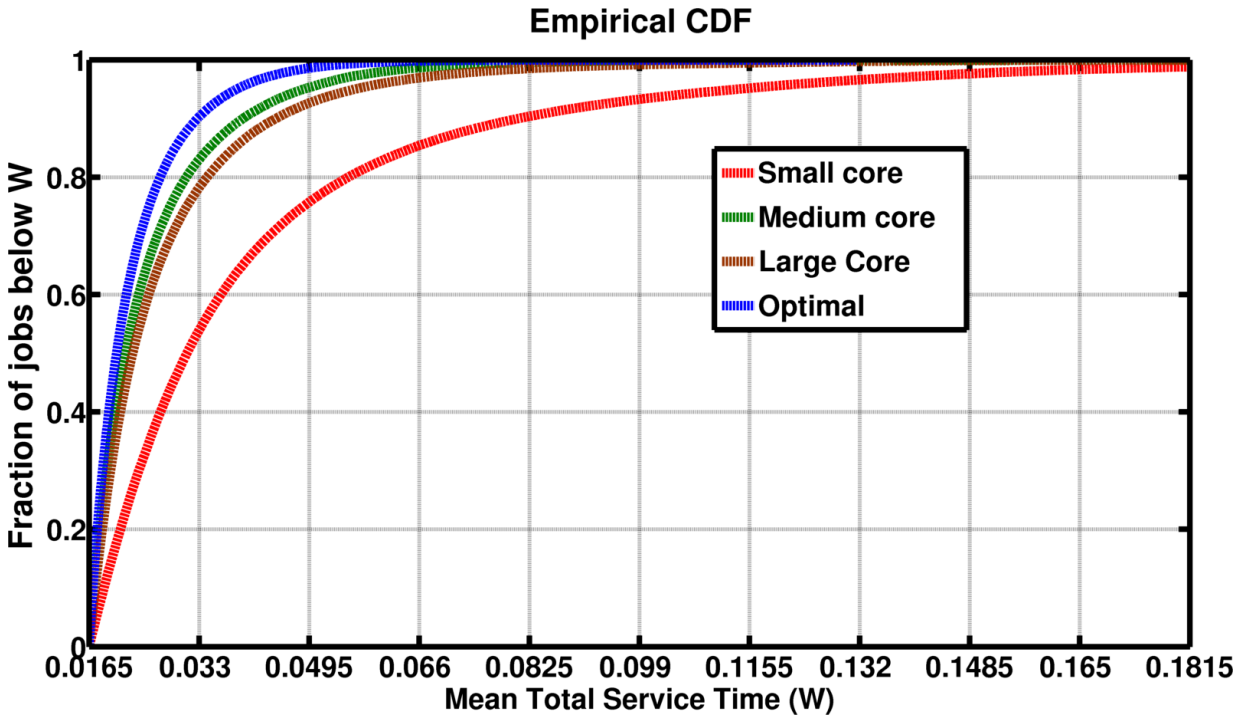
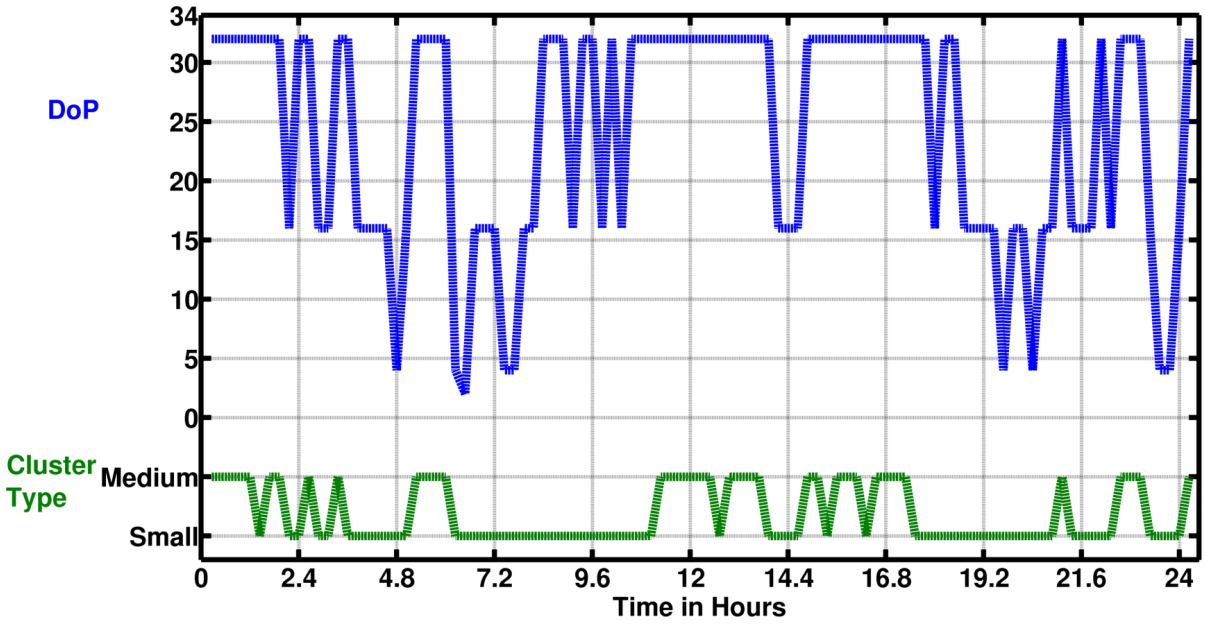


Figure 5.6: Cumulative distribution function of total time spent in the system for Barnes benchmark

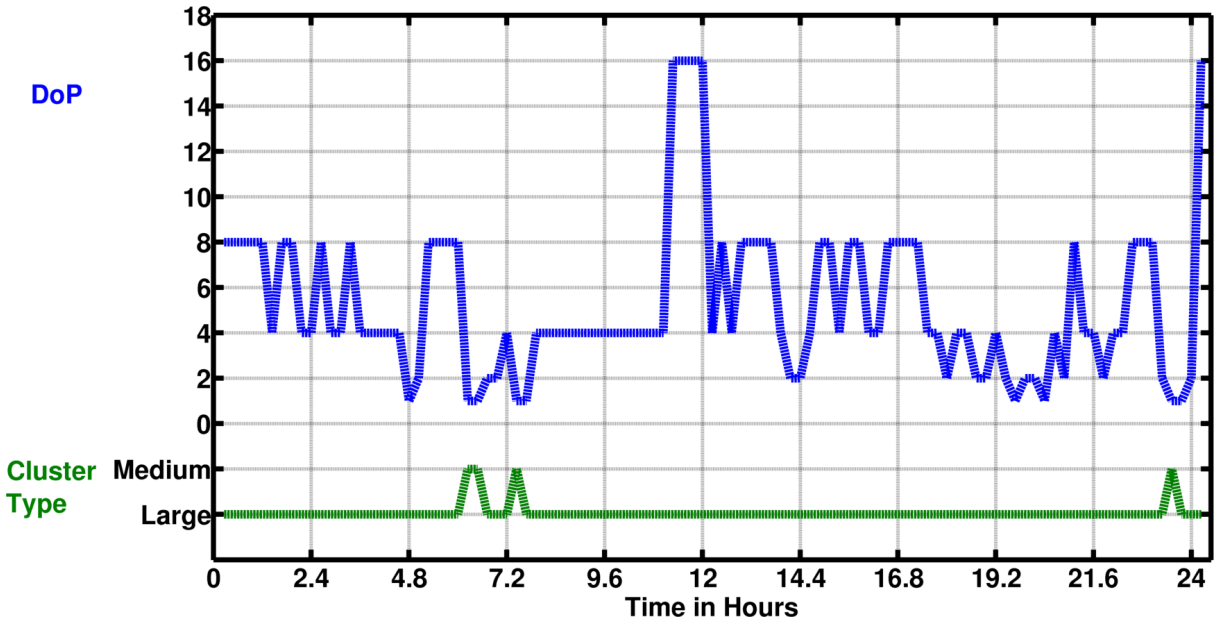
Raytrace benchmark as show in Figure 5.7b .Typically, there is a discrepancy between the execution time of a single job (milliseconds to seconds) and the variation in arrival rate (over minutes or hours). In Figure 5.7a, the mean number of jobs that execute per change in cluster type is more than 20,000. Thus, the performance overhead of cluster migration is amortized over the relatively long period of time spent in each cluster. As another example, Figure 5.7b shows the optimal DoP and optimal cluster type as a function of time for the Raytrace benchmark. We can see that there are relatively few migrations as with the Radix benchmark.

To further understand the overhead of cluster migration, we ran an experiment on an architecture with two *identical* medium clusters, and migrated jobs from one cluster to another at uniform intervals of time varying from 22 to 110 migrations per minute. The overhead with respect to the scenario without migrations is shown in Figure 5.8. Even with these relatively frequent migrations (relative to variations in job arrival rate), we find that the overheads of migration are within 0.4%.

These jobs were from the Radix benchmark with jobs arriving at the maximum sus-



(a) Radix



(b) Raytrace

Figure 5.7: Graph showing the variation in optimal DoP and cluster type for (a) Radix (b) Raytrace benchmarks under varying arrival rate.

tainable arrival rate. The DoP in this case, as expected was one. 22 migrations per minute roughly translates into a migration every 100 jobs completed. It can be seen that the overhead is always below 0.5%.

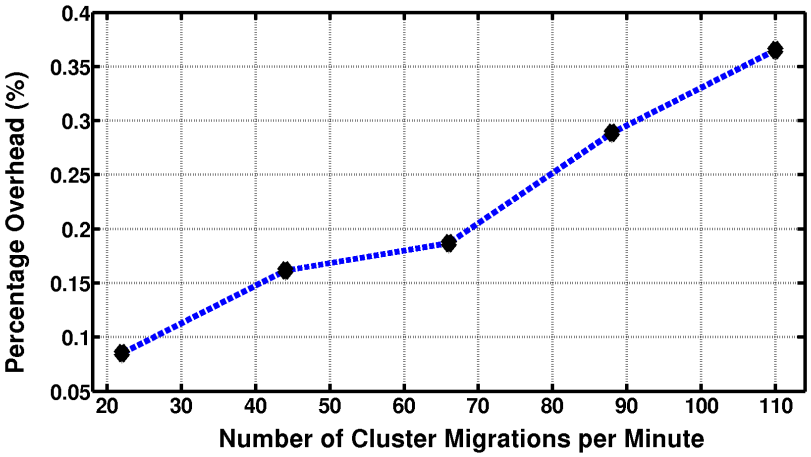


Figure 5.8: Graph comparing service time with number of migrations per minute against a baseline, where no migrations happen.

5.2.1 Comparison with Dim Silicon architecture

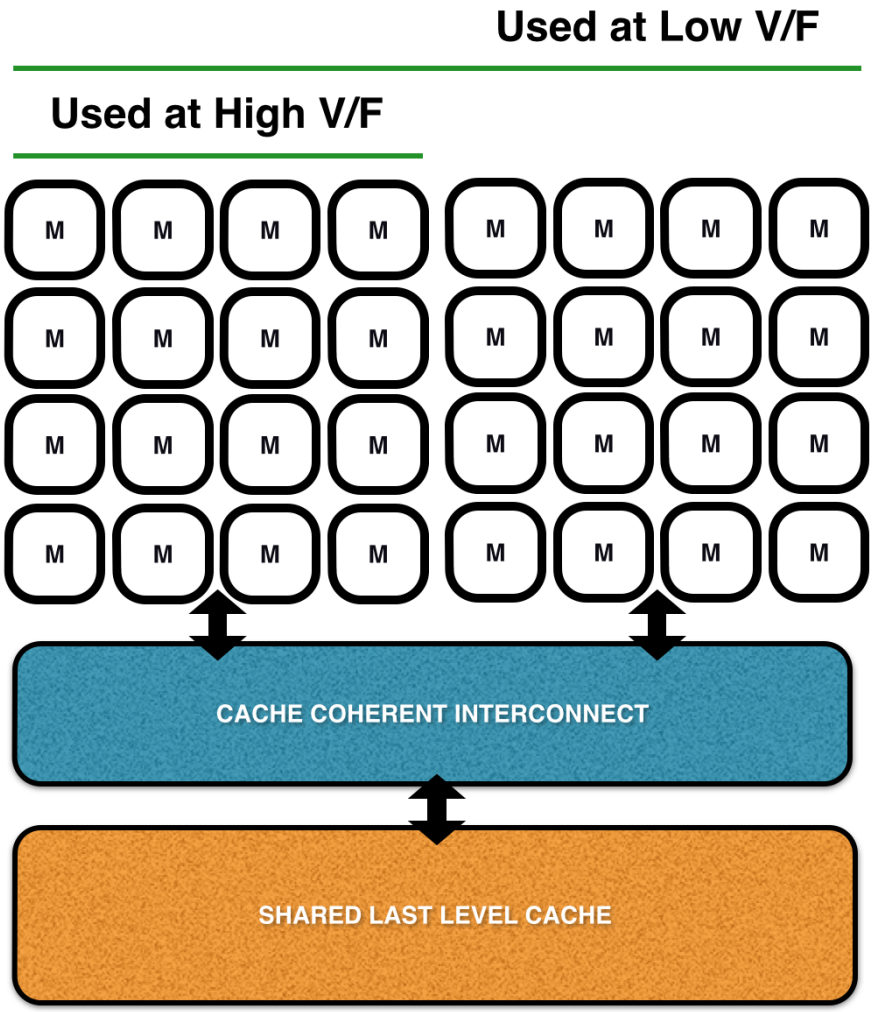


Figure 5.9: An example heterogeneous Dim Silicon architecture

We now compare the clustered asymmetric architecture discussed so far with a homogeneous dark silicon CMPs using dynamic voltage frequency scaling (DVFS). Recent work has looked at using DVFS to have a few of cores turned on at a high voltage/frequency (V/F) level or have more cores turned on at a lower V/F level. The later mode of operation referred to as "dim silicon".We compare the asymmetric clustered architecture with a dim silicon architecture consisting of 64 medium cores and 2 V/F levels. In the high V/F level

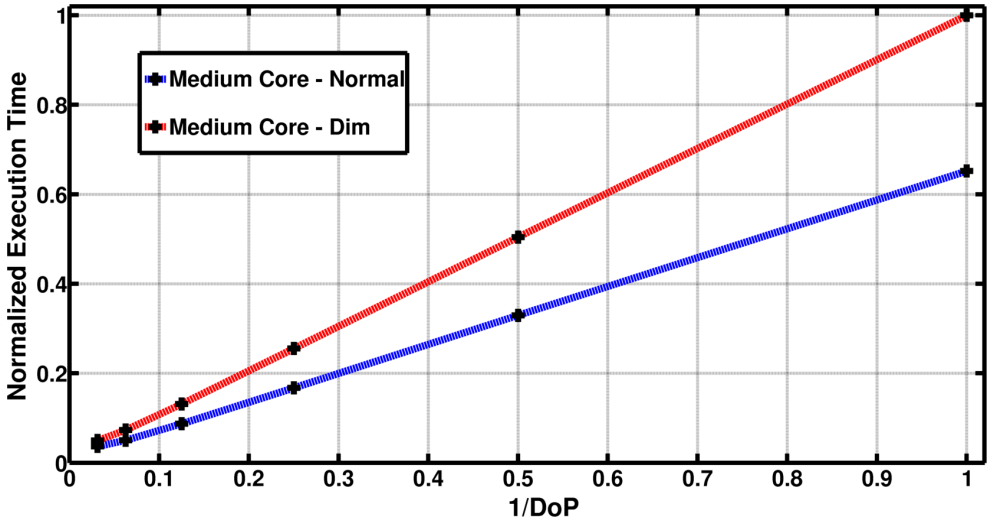


Figure 5.10: Normalized execution time vs degree of parallelism for the Radix benchmark in both V/F levels

we have 32 cores which can be turned on and in the low V/F level all 64 medium cores can be turned on. Remember that 32 medium cores at the high V/F level take up the entire power budget of 85W. Table 5.1 shows the V/F pairs used.

Voltage(V)	Frequency(GHz)
0.7	4.5
0.6	2.93

Table 5.1: V/F pairs used for the homogeneous dark silicon architecture. Values are shown for 11 nm technology node using scaling factors indicated by [3]

Figure 5.10 shows the graph for normalized execution time versus degree of parallelism for the radix benchmark for the two V/F levels.

We perform the comparison with two benchmarks from each benchmark suite over the real world arrival rate curve used before. Here we scale the real world arrival rate curve to maximum sustainable arrival rate from the asymmetric clustered architecture. Figure 5.11 plots the mean total service time (normalized to the mean total service time of the dim silicon architecture with oracular knowledge of the arrival rate) for the chosen benchmarks for the different scheduling approaches mentioned before. The clustered asymmetric architecture does 25% better than the dim silicon architecture with oracular information of the arrival rate and 21% better when the arrival is predicted using the naive predictor.

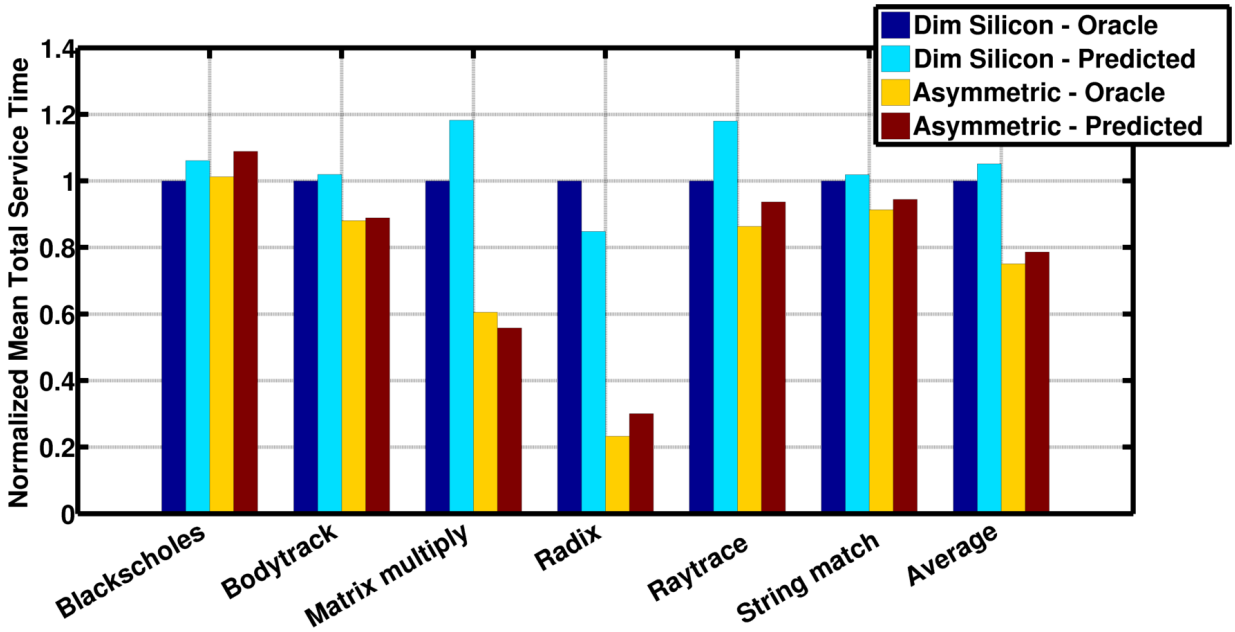


Figure 5.11: The percentage improvement in the mean total service time over the chosen benchmarks with oracular knowledge of the arrival rate and using the naive prediction of the arrival rate with baseline as the Dim Silicon architecture with oracular knowledge of the arrival rate

All though the clustered asymmetric architecture does better on average when compared to dim silicon, it is not the case for the Blackscholes benchmark. With respect to the Blackscholes benchmark, the dim silicon architecture outperforms the clustered asymmetric architecture by 1% with oracular knowledge of the arrival rate and by 9% when using the prediction mechanism.

Changes in maximum sustainable arrival rate.

In the previous experiments the real-world arrival rate curve has been scaled to the maximum sustainable arrival rate from the clustered asymmetric architecture. For some benchmarks, the dim silicon architecture showed an increase in maximum sustainable arrival rate, with 2.85% increase on average. In some cases, like Raytrace the maximum sustainable arrival rate increased by as much as32.39%. Results for all the benchmarks are tabulated in table 5.2

Benchmark	% change in λ_{max}
Blackscholes	7.70
Bodytrack	4.54
Matrix Multiply	-1.34
Radix	-17.37
Raytrace	32.39
String Match	-8.77
Average	2.85

Table 5.2: Percentage change in maximum sustainable arrival rate of dim silicon architecture over clustered asymmetric architecture.

Chapter 6

Conclusions

In this thesis we have shown, both analytically and empirically, that the optimal degree of parallelism (DoP) for multi-threaded applications executing on multi-core servers in a data-center changes with job arrival rate. In addition, for asymmetric multi-core processors with multiple clusters, the optimal cluster type also varies with job arrival rate. Based on these observations, we have proposed a job arrival rate run-time scheduler that makes optimal cluster migration and DoP selection decisions so as to minimize mean service time within a power budget. Experimental results that compared to a baseline static scheduler, the proposed scheduling mechanism can improve mean service time by 34%, averaged over a wide range of benchmark applications.

Chapter 7

Future Work

For future work, we plan to focus on the following

Arrival Rate Prediction. The arrival rate predictor used by us was only 8.7% worse than with the oracular knowledge when optimally selecting the DoP and only 8.1% worse than the oracular knowledge when optimally selecting both DOP and cluster migration. This may not be true for all arrival rate curves. Hence, a better prediction mechanism would certainly improve the performance.

Run-time Characterization of Workload properties. We assume pre-characterized information about workloads and their service times on each of the clusters. This might not be feasible. Characterizing the workload during run-time will help run different types workloads in the future.

Migration Policies. When the need arises to migrate to a different cluster, we use a naive policy of waiting for waiting for all current jobs to finish before switching clusters. As stated in the results the overhead of migration is not too much but it can be improved with better migration policies.

Dim Silicon Architecture. Though the dim silicon architecture was worse than the clustered asymmetric on average. It did provide benefits in terms of havinng higher maximum sustainable arrival rate. It might be useful to pursue a hybrid approach.

References

[1] B. Raghunathan, Y. Turakhia, S. Garg, and D. Marculescu, "Cherry-picking: exploiting process variations in dark-silicon homogeneous chip multi-processors," in *Proceedings of the Conference on Design, Automation and Test in Europe*.

[2] B. Raghunathan and S. Garg, "Job arrival rate aware scheduling for asymmetric multicore servers in the dark silicon era," to appear in Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2014 IEEE/ACM/IFIP International Conference on, 2014.

[3] H. Esmaeilzadeh, E. Blem, R. St Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*, 2011.

[4] B. Jeff, *Advances in big.LITTLE Technology for Power and Energy Savings*, September 2012. [Online]. Available: http://www.arm.com/files/pdf/Advances_in_big.LITTLE_Technology_for_Power_and_Energy_Savings.pdf

[5] R. Almeida, B. Mozafari, and J. Cho, "On the evolution of wikipedia," in *International Conference on Weblogs and Social Media*, 2007.

[6] L. A. Barroso and U. Holzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, 2007.

[7] H.-D. Cho, C. Kisuk, and T. Kim, *Benefits of the big.LITTLE Architecture*, February 2012. [Online]. Available: http://www.samsung.com/global/business/semiconductor/minisite/Exynos/data/benefits.pdf

[8] N. Leavitt, "Is cloud computing really ready for prime time?" *Computer*, vol. 42, 2009.

[9] D. Hamilton, *SAP to Create Cloud Computing Solutions for Specific Industries*, June 2014. [Online]. Available: http://www.thewhir.com/web-hosting-news/sap-create-cloud-computing-solutions-specific-industries

[10] G. E. Moore, "Cramming more components onto integrated circuits," *Proceedings of the IEEE*, vol. 86, 1998.

[11] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc, "Design of ion-implanted MOSFET's with very small physical dimensions," *Solid-State Circuits, IEEE Journal of*, vol. 9, no. 5, 1974.

[12] G. Venkatesh, J. Sampson, N. Goulding, S. Garcia, V. Bryksin, J. Lugo-Martinez, S. Swanson, and M. B. Taylor, "Conservation cores: Reducing the energy of mature computations," in *Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems*. ACM, 2010.

[13] M. B. Taylor, "Is dark silicon useful?: harnessing the four horsemen of the coming dark silicon apocalypse," in *Proceedings of the 49th Annual Design Automation Conference*, 2012.

[14] W. Huang, K. Rajamani, M. R. Stan, and K. Skadron, "Scaling with design constraints: Predicting the future of big chips," *IEEE Micro*, vol. 31, 2011.

[15] Y. Turakhia, B. Raghunathan, S. Garg, and D. Marculescu, "HaDeS: Architectural synthesis for heterogeneous dark silicon chip multi-processors," in *Proceedings of the 50th Annual Design Automation Conference*, 2013.

[16] E. Rahm, "Dynamic load balancing in parallel database systems," in *Euro-Par'96 Parallel Processing*, 1996.

[17] A. Raman, H. Kim, T. Oh, J. W. Lee, and D. I. August, "Parallelism orchestration using DoPE: the degree of parallelism executive," in *ACM SIGPLAN Notices*, vol. 46, no. 6, 2011.

[18] J. Li and J. F. Martinez, "Dynamic power-performance adaptation of parallel computation on chip multiprocessors," in *The Twelfth International Symposium on High-Performance Computer Architecture, 2006*, 2006.

[19] H. Sasaki, T. Tanimoto, K. Inoue, and H. Nakamura, "Scalability-based manycore partitioning," in *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*.

[20] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas, "Single-ISA heterogeneous multi-core architectures for multithreaded workload performance," in *ACM SIGARCH Computer Architecture News*, vol. 32, no. 2, 2004.

[21] J. H. Ahn, S. Li, O. Seongil, and N. P. Jouppi, "McSimA+: A manycore simulator with application-level+ simulation and detailed microarchitecture modeling," in *Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on*.

[22] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer, "Scheduling heterogeneous multi-cores through performance impact estimation (PIE)," in *ACM SIGARCH Computer Architecture News*, June 2012.

[23] T. S. Muthukaruppan, M. Pricopi, V. Venkataramani, T. Mitra, and S. Vishin, "Hierarchical power management for asymmetric multi-core in dark silicon era," in *Proceedings of the 50th Annual Design Automation Conference*, 2013.

[24] F. S. Hillier, *Intro To Operations Research*. Tata McGraw-Hill Education, 1995.

[25] G. M. Amdahl, "Validity of the single processor approach to achieving large-scale computing capabilities," in *Proceedings of the April 18-20, 1967, spring joint computer conference*, 1967.

[26] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, 2009.

[27] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011.

[28] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: characterization and methodological considerations," in *ACM SIGARCH Computer Architecture News*, 1995.

[29] C. Bienia and K. Li, "PARSEC 2.0: A new benchmark suite for chip-multiprocessors," in *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, 2009.

[30] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis, "Evaluating mapreduce for multi-core and multiprocessor systems," in *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on*, Feb 2007.

[31] V. Jalaparti, H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Bridging the tenant-provider gap in cloud services," in *Proceedings of the Third ACM Symposium on Cloud Computing*.

[32] D. Meisner, J. Wu, and T. F. Wenisch, "Bighouse: A simulation infrastructure for data center systems," in *Performance Analysis of Systems and Software (ISPASS), 2012 IEEE International Symposium on*, 2012.

[33] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "The case for evaluating mapreduce performance using workload suites," in *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on*.