DISSERTATION

HETEROGENEOUS COMPUTING ENVIRONMENT CHARACTERIZATION AND THERMAL-AWARE SCHEDULING STRATEGIES TO OPTIMIZE DATA CENTER POWER CONSUMPTION

Submitted by

Abdulla Al-Qawasmeh

Department of Electrical and Computer Engineering

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Summer 2012

ABSTRACT

HETEROGENEOUS COMPUTING ENVIRONMENT CHARACTERIZATION AND THERMAL-AWARE SCHEDULING STRATEGIES TO OPTIMIZE DATA CENTER POWER CONSUMPTION

Many computing systems are heterogeneous both in terms of the performance of their machines and in terms of the characteristics and computational complexity of the tasks that execute on them. Furthermore, different tasks are better suited to execute on specific types of machines. Optimally mapping tasks to machines in a heterogeneous system is, in general, an NP-complete problem. In most cases, heuristics are used to find near-optimal mappings. The performance of allocation heuristics can be affected significantly by factors such as task and machine heterogeneities. In this thesis, different measures are identified to be used in quantifying the heterogeneity of HC systems and the correlation between the performance of the heuristics and these measures is shown.

The power consumption of data centers has been increasing at a rapid rate over the past few years. Motivated by the need to reduce the power consumption of data centers, many researchers have been investigating methods to increase the energy efficiency in computing at different levels: chip, server, rack, and data center. Many of today's data centers experience physical limitations on the power needed to run the data center. The first problem that is studied in this thesis is maximizing the performance of a data center that is subject to total power consumption and thermal constraints. A power model for a data center that includes power consumed in both Computer Room Air Conditioning (CRAC) units and compute nodes is considered. The approach in this thesis quantifies the performance of the data center as the total reward collected from completing tasks in a workload by their individual deadlines. The second problem that is studied in this research is how to minimize the power consumption in a data center while guaranteeing that the overall performance does not drop below a specified threshold. For both problems, novel optimization techniques for assigning the performance states of compute cores at the data center level to optimize the operation of the data center are developed. The assignment techniques are divided into two stages. The first stage assigns the P-states of cores, the desired number of tasks per unit time allocated to a core, and the outlet CRAC temperatures. The second stage assigns individual tasks as they arrive at the data center to cores so that the actual number of tasks per unit time allocated to a core approaches the desired number set by the first stage.

ACKNOWLEDGMENTS

I thank my advisors Professors Siegel and Maciejewski for their valuable guidance throughout my Ph.D. studies. I also thank Jerry Potter, Greg Pfister, Sudeep Pasricha, Paul Maxwell, Luis Briceño, Arnold Rosenberg, Ron Chiang, Bhavish Khemka, Mark Oxley, Nishit Kapadia, Ryan Friese, Dalton Young, Timothy Hansen, Rodney Roberts, Jay Smith and everybody at Colorado State University for their valuable comments and help.

DEDICATION

To may my family for their support, prayers, and love.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Heterogeneous Computing (HC) systems studied in this thesis are ones that consist of a set of different machines that have varying capabilities. These machines are used to execute a set of heterogeneous tasks that vary in their computational complexity. Heuristics are used to assign tasks to machines in an HC system to optimize some objectives. The objectives considered in this research are the makespan (in Chapter 2), the power consumption (in Chapter 4), and the reward collected from completing tasks by their individual deadline (in Chapter 4).

Chapter 2 presents three statistical measures that can be used to quantify the heterogeneity of computing systems. These measures are the coefficient of variation (COV), skewness (third moment), and kurtosis (fourth moment). The coefficient of variation has been used in [AlS00] to generate simulated heterogeneous environments. We show how to use COV as one measure for quantifying the heterogeneity of *existing* heterogeneous environments. The importance of using the three statistical measures is demonstrated through simple task allocation examples and simulations. We show the correlation between the statistical measures and the performance of some greedy allocation heuristics. This correlation can be used to make better decisions about the most appropriate heuristic for a given HC system and the potential performance of an HC system.

In Chapter 3, we identify three properties that heterogeneity measures should have and propose measures that have these properties. These measures are the machine performance homogeneity (MPH), task easiness homogeneity (TEH), and task-machine affinity (TMA). To illustrate the intuition behind MPH, TEH, and TMA, we show some simple HC systems that consist of two machines and two task types and where they fall within the range of all possible values of the three measures. We use information from the integer and floating point SPEC benchmarks (SPEC CINT2006Rate and SPEC CFP2006Rate [Spe06]) to construct HC systems. These systems illustrate how environments constructed from real world task types and machines can have widely varying values for each of the measures proposed in Chapter 3.

The power consumption of data centers has been increasing at a rapid rate over the past few years. Further, many of today's data centers experience physical limitations on the power needed to run the data center. The first problem that we study in Chapter 4 is maximizing the performance of a data center that is subject to total power consumption and thermal constraints. We consider a power model for a data center that includes power consumed in both Computer Room Air Conditioning units and compute nodes. Our approach quantifies the performance of the data center as the total reward collected from completing tasks in a workload by their individual deadlines. The second problem that we study in this research is how to minimize the power consumption in a data center while guaranteeing that the overall performance does not drop below a specified threshold. For both problems, we develop novel optimization techniques for assigning the performance states of compute cores at the data center level to optimize the operation of the data center. The assignment techniques are divided into two stages. The first stage assigns the P-states of cores, the desired number of tasks per unit time allocated to a core, and the outlet CRAC temperatures. The second stage assigns individual tasks as they arrive at the data center to cores so that the actual number of tasks per unit time allocated to a core approaches the desired number set by the first stage.

Chapter 2

Statistical Measures for Quantifying Task and Machine Heterogeneities

2.1. Introduction

We study heterogeneous computing (HC) systems that consist of a set of different machines that have varying capabilities. These machines are used to execute a set of heterogeneous tasks that vary in their computational complexity. Many of today's supercomputers are heterogeneous, e.g., the Extreme Scale Systems Center (ESSC) at Oak Ridge National Laboratory (ORNL). Furthermore, many of the homogeneous systems today may become heterogeneous in the future by adding new processing units with different capabilities than the existing ones.

The Estimated Time to Compute (ETC) each task on each machine in an HC system is arranged in an ETC matrix, where entry ETC(*i*, *j*) is the estimated execution time of task *i* on machine *j* when executed alone. The assumption of such ETC information is a common practice in resource allocation research (e.g., [BaS01, DhA02, GhY93, KaA98, KhP93, SiY96, XuN01]). An ETC matrix for a given HC system can be obtained from user supplied information, experimental data, or task profiling and analytical benchmarking [AlB05, GhY93, KhP93, XuN01].

Machine heterogeneity is the degree to which the execution time of a given task varies

A preliminary version of portions of this chapter appeared in [AlM11b].

for different machines (the variation along the same row of an ETC matrix). Analogously, task heterogeneity is the degree to which the execution times of different tasks vary for the same machine (the variation along the same column) in the ETC matrix.

In an HC system, tasks should be mapped (allocated) to the available machines in a way that optimizes some performance objective (e.g., [BaS01, BaV01, BrS01, BrS08, BuP97, ChM10, KiH06, MaA99, MeS07, MiF00, WuS00]). Mapping tasks to machines in HC systems has been shown to be, in general, an NP-complete problem [Cof76, Fer89, IbK77]. Hence, many heuristics have been developed for allocating tasks to machines in HC systems. The performance of allocation heuristics and the HC system is affected by several factors one of which is the level of machine heterogeneity [BrS01, ChM10]. Therefore, quantifying the heterogeneity of a given environment will allow the selection of a heuristic that is the most appropriate.

In most previous work (e.g., [AlS00, BrS01, CaJ09, EsA09, HuY09, KhA06]), either the range of the execution time values or their coefficient-of-variation (COV) was used as a measure of the heterogeneity to generate ETC matrices for simulation studies. These measures do not completely represent the possible variation in heterogeneity. For example, many ETC matrices with the same COV can have other statistical properties that are vastly different and that may be highly correlated with the performance of a mapping heuristic. Furthermore, these measures were intended for quantifying the heterogeneity of existing HC systems.

The contributions of this chapter are the use of: (1) different statistical measures to quantify task and machine heterogeneities for existing HC systems, (2) simple mapping examples and synthetic data analysis that demonstrate the importance of these measures, and (3) regression trees to predict the most appropriate heuristic for a given HC system based on its heterogeneity. The regression trees demonstrate the importance of the heterogeneity measures in significantly reducing the error in predicting the most appropriate heuristic.

The remainder of this chapter is organized as follows. The procedures for each of the studied heuristics are given in Section 2.2. In Section 2.3, we describe different statistical measures for quantifying heterogeneity. Heuristic selection based on the heterogeneity measures is discussed in Section 2.4. Synthetic data analysis to demonstrate heuristic selection based on heterogeneity is given in Section 2.5. Section 2.6 discusses related work. Conclusions are given in Section 2.7.

2.2. Heuristics

The task mapping problem that we study in this chapter is a static one (i.e., task mapping decisions are made before any task is executed). Further, there are no inter-task dependences. This mapping problem has been studied widely (e.g., [ArH98, IbK77, WuS00, WuS01]). The makespan of a task mapping is the latest finish time among all machines, where a lower makespan is better. We study five heuristics to derive task mappings that minimize the makespan: Min-Min [IbK77], Max-Min [IbK77], Sufferage [MaA99], MCT (minimum completion time) [ArH98], and KPB (*k*-percent best) [MaA99]. The performance of the heuristics is quantified by the makespan.

The Min-Min heuristic passes through the unmapped tasks twice. In the first pass, for each task, it finds the machine that gives the *minimum* completion time. In the second pass, it finds the task with the *minimum* overall completion time and maps it to the minimum completion time machine identified in the first pass. The two passes are repeated until all tasks are assigned. The pseudocode of the Min-Min heuristic is given in Figure 2.1. The Max-Min heuristic is similar to Min-Min except that in the second pass instead of mapping the task that has the

```
do the following steps while there are unmapped tasks
(1) for each unmapped task, determine the machine that gives the task its *minimum*
    completion time.
(2) among the task-machine pairs determined in (1), map the task that has the *minimum*
    overall completion time to the corresponding machine.
(3) update the ready times of each machine
```

Figure 2.1. The pseudocode of the Min-Min heuristic.

minimum completion time it maps the task that has the *maximum* completion time (i.e., Max-Min starts by assigning the tasks with longer execution times).

The sufferage heuristic attempts to minimize the makespan by assigning every task to its best machine, but when multiple tasks want the same machine it gives preference to the task that would "suffer" the most if it were not assigned to that machine. The sufferage value of a given task is the difference between its completion time on the machine with the second earliest completion time and the completion time on the machine with the earliest completion time. The pseudocode of the sufferage heuristic is given in Figure 2.2.

As the name implies, the MCT heuristic loops thought all of the tasks in arbitrary order and assigns each task to the machine that gives the minimum completion time for that task. The KPB heuristic also loops through the tasks in arbitrary order. However, for a given task, KPB considers a subset of machines for mapping that task. The subset is formed by picking the *k*-percent of all of the machines that have the best (lowest) execution time for that task. Within the *k*-percent best machines the task is assigned to the machine with the minimum completion time. The *k* value is a parameter that can be set by the user. For the purpose of illustration, we used a *k* value of 0.75.

```
do the following while there are still unmapped tasks:
(1) for each machine find the set of tasks that have their minimum completion time on
    this machine:
      (a) if the size of the set is one, then assign the corresponding task to the machine.
      (b) if the size of the set is greater than one, then assign the task with the highest
          sufferage value to the corresponding machine.
(2) update the ready times for all the machines.
```

Figure 2.2. The pseudocode of the sufferage heuristic.

2.3. Measuring Heterogeneity

2.3.1. Introduction

In this section, we illustrate how three statistical measures are used to quantify the heterogeneity of an HC system represented by an ETC matrix. These statistical measures are: (a) coefficient of variation, (b) skewness, and (c) kurtosis.

The following variables will be used in the calculation of each of the statistical measures: $\underline{T}$ is the number of tasks to be mapped, $\underline{M}$ is the number of machines in the system, and $\underline{\mu_i^{(\mathrm{t})}}$ is the mean ETC of task i over all machines, given by $\mu_i^{(\mathrm{t})} = \frac{1}{M}\sum_{j=1}^{M}\mathrm{ETC}(i,j)$. The mean ETC of all tasks on machine j, $\underline{\mu_j^{(\mathrm{m})}}$, is given by $\mu_j^{(\mathrm{m})} = \frac{1}{T}\sum_{i=1}^{T}\mathrm{ETC}(i,j)$. The standard deviation of the ETC of task i over all machines, $\underline{\sigma_i^{(\mathrm{t})}}$, is given by $\sigma_i^{(\mathrm{t})} = \sqrt{\frac{1}{M}\sum_{j=1}^{M}(\mathrm{ETC}(i,j) - \mu_i^{(\mathrm{t})})^2}$. The standard deviation of the ETC of all tasks on machine j, $\underline{\sigma_j^{(\mathrm{m})}}$, is given by $\sigma_j^{(\mathrm{m})} = \sqrt{\frac{1}{T}\sum_{i=1}^{T}(\mathrm{ETC}(i,j) - \mu_j^{(\mathrm{m})})^2}$.

2.3.2. Coefficient of Variation

The COV has been used in [AlS00] to generate ETC matrices with different task and machine heterogeneities. For a set of values with standard deviation σ and mean μ, the COV is given by $\mathrm{COV} = \frac{\sigma}{\mu}$. Let $\underline{V_i^{(\mathrm{t})}}$ for task i be the COV over all machines, given by $V_i^{(\mathrm{t})} = \frac{\sigma_i^{(\mathrm{t})}}{\mu_i^{(\mathrm{t})}}$. Let $\underline{V_j^{(\mathrm{m})}}$ for machine j be the COV over all tasks, given by $V_j^{(\mathrm{m})} = \frac{\sigma_j^{(\mathrm{m})}}{\mu_j^{(\mathrm{m})}}$. Task heterogeneity as measured by the Average Task COV (ATC) is given by $\mathrm{ATC} = \left[\sum_{j=1}^{M} V_j^{(\mathrm{m})}\right] / M$. Machine heterogeneity as measured by the Average Machine COV (AMC) is given by $\mathrm{AMC} = \left[\sum_{i=1}^{T} V_i^{(\mathrm{t})}\right] / T$.

Although both ATC and AMC quantify the variation of the execution time values, they do not indicate whether most of the values are less than or greater than the mean, and whether the variation is caused by many values having an average deviation from the mean, or a small number of values having a large deviation from the mean. These are quantified by skewness and kurtosis, respectively. Sections 2.4 and 2.5 describe how skewness and kurtosis may have an effect on the performance of heuristics. Using these measures we may be able to select a more appropriate heuristic for an HC system.

2.3.3. Skewness

The skewness of a set of values measures the degree of asymmetry of the values over the mean [PrF88]. Skewness corresponds to the third moment of a distribution. Positive skewness means that most of the values are below the mean and negative skewness means that most of the values are greater than the mean.

Let $S_i^{(\mathrm{t})}$ for task i be the skewness over all machines, given by $S_i^{(\mathrm{t})} = \left[\frac{1}{M} \sum_{j=1}^{M} (\mathrm{ETC}(i,j) - \mu_i^{(\mathrm{t})})^3 \right] / (\sigma_i^{(\mathrm{t})})^3$. Let $S_j^{(\mathrm{m})}$ for machine j be the skewness over all tasks, given by $S_j^{(\mathrm{m})} = \left[\frac{1}{T} \sum_{i=1}^{T} (\mathrm{ETC}(i,j) - \mu_j^{(\mathrm{m})})^3 \right] / (\sigma_j^{(\mathrm{m})})^3$. Task heterogeneity as measured by the Average Task Skewness (ATS) is given by $\mathrm{ATS} = \left[\sum_{j=1}^{M} S_j^{(\mathrm{m})} \right] / M$. Machine heterogeneity as measured by the Average Machine Skewness (AMS) is given by $\mathrm{AMS} = \left[\sum_{i=1}^{T} S_i^{(\mathrm{t})} \right] / T$.

2.3.4. Kurtosis

The kurtosis of a set of values measures the extent to which the deviation is caused by a small number of values having extreme deviations from the mean versus a large number of values having modestly sized deviations [PrF88]. Kurtosis corresponds to the fourth moment of a distribution. Higher values of kurtosis indicate that the standard deviation is caused by fewer values having extreme deviations. The definition of kurtosis that we use is the excess kurtosis [PrF88]. Excess kurtosis equals kurtosis minus 3. This makes the excess kurtosis of the Gaussian (normal) distribution equal to 0.

Let $\underline{K_i^{(\mathrm{t})}}$ for task i be the kurtosis over all machines, given by $K_i^{(\mathrm{t})} = \left[\left(\frac{1}{M} \sum_{j=1}^{M} (\mathrm{ETC}(i,j) - \mu_i^{(\mathrm{t})})^4 \right) / (\sigma_i^{(\mathrm{t})})^4 \right] - 3$. Let $\underline{K_j^{(\mathrm{m})}}$ for machine j be the kurtosis over all tasks, given by $K_j^{(\mathrm{m})} = \left[\left(\frac{1}{T} \sum_{i=1}^{T} (\mathrm{ETC}(i,j) - \mu_j^{(\mathrm{m})})^4 \right) / (\sigma_j^{(\mathrm{m})})^4 \right] - 3$. Task heterogeneity as measured by the

Average Task Kurtosis (ATK) is given by $\mathrm{ATK} = \frac{1}{M}\sum_{j=1}^{M} K_j^{(\mathrm{m})}$. Machine heterogeneity as measured by the Average Machine Kurtosis (AMK) is given by $\mathrm{AMK} = \frac{1}{T}\sum_{i=1}^{T} K_i^{(\mathrm{t})}$.

2.4. Heuristic Selection

2.4.1. Overview

This section illustrates how the heterogeneity measures can be used to make heuristic selection decisions based on the heterogeneity of an HC system. Section 2.4.2 illustrates the correlation that ATS and ATK may have with the performance of Max-Min and Min-Min heuristics by using simple mapping examples. Heuristic selection decisions based on a single heterogeneity measure and based on multiple heterogeneity measures are illustrated in Sections 0 and 0, respectively.

2.4.2. Simple Mapping Examples

The Min-Min and Max-Min heuristics have been studied widely (e.g., [AlK08, BrS01, BrS08, DiC01, GhM05, IbK77, JiL05, KaU07, MaA99, WuS00]). Therefore, in this section, we selected these two heuristics to illustrate how skewness and kurtosis may affect the performance of the two heuristics using some simple task mapping examples.

Figure 2.3 shows a scenario in which the Max-Min heuristic outperforms the Min-Min heuristic for high values of ATS. The ETC shown in Figure 2.3 has a positive ATS value of 0.62. A pictorial representation of the assignments made by each heuristic is given in the figure. The makespan of the mapping produced by the Min-Min heuristic is 44 and the makespan of the mapping produced by Max-Min is 36.

	m_1	m_2
t_1	6	10
t_2	14	8
t_3	40	36

AMC = 0.19
ATC = 0.72
AMS = 0
ATS = 0.62
AMK = −2
ATK = −1.5

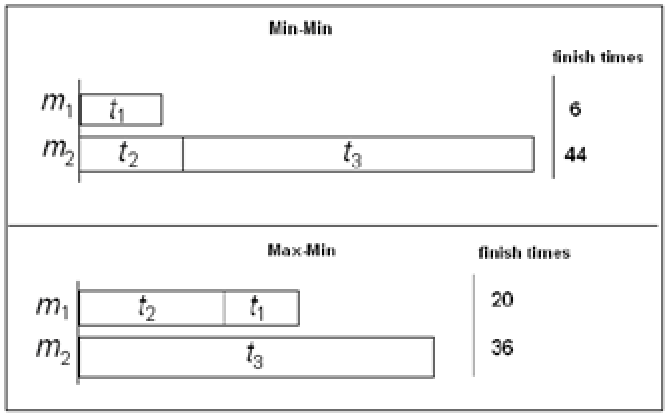


Figure 2.3. An example that illustrates a situation where the Max-Min heuristic outperforms the Min-Min heuristic for an ETC matrix with high positive ATS. Shown in the figure are the ETC matrix, the values of the statistical measures for the matrix, and a pictorial representation of the mappings produced by the Max-Min and the Min-Min heuristics.

Although Min-Min performs better than Max-Min in most cases (e.g., [BrS01, BrS08]), Max-Min usually has better performance than Min-Min when there are many shorter tasks than there are longer ones, i.e., the corresponding ETC matrix has high positive task skewness. This is because Max-Min starts by assigning the longer tasks to their best machine.

An example where Min-Min outperforms Max-Min for an ETC matrix with negative ATS is given in Figure 2.4. The ETC in the figure has a negative ATS value of –0.23. A pictorial representation of the mapping produced by each heuristic is given in the figure. The makespan of the mapping produced by Min-Min is 46 and the makespan of the mapping produced by Max-Min is 56.

To show how kurtosis of an ETC matrix may impact the performance of the two heuristics, we compare the examples given in Figures 2.5 and 2.6. The ETC matrices in both figures have an ATS value that is 0 or close to 0. Therefore, using the ATS will result in the same heuristic selection decision. However, using the ATK, better decisions can be made.

The ETC matrix in the Figure 2.5 has a high ATK value of 0.92 (compared to the kurtosis of the normal distribution which is 0 and the uniform distribution which is –1.2). In Figure 2.5,

	m_1	m_2
t_1	6	12
t_2	56	24
t_3	40	35

AMC = 0.27
ATC = 0.51
AMS = 0
ATS = − 0.23
AMK = −2
ATK = −1.5

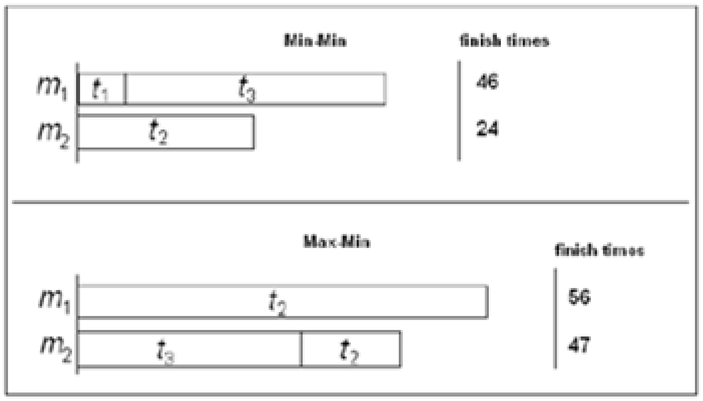


Figure 2.4. An example that illustrates a situation where the Min-Min heuristic outperforms the Max-Min heuristic for an ETC matrix with negative ATS. Shown in the figure are the ETC matrix, the values of the statistical measures for the matrix, and a pictorial representation of the mappings produced by the Min-Min and the Max-Min heuristics.

	m_1	m_2
t_1	3	5
t_2	55	53
t_3	49	47
t_4	50	52
t_5	54	56
t_6	45	51
t_7	54	50
t_8	105	93

AMC = 0.06
ATC = 0.47
AMS = 0
ATS = 0
AMK = −2
ATK = 0.92

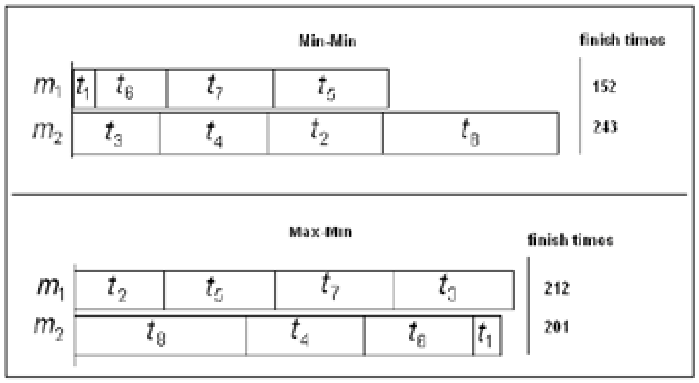

Min-Min
finish times
m_1 t_1 t_6 t_7 t_5
m_2 t_3 t_4 t_2 t_8
152
243
Max-Min
finish times
m_1 t_2 t_5 t_7 t_3
m_2 t_8 t_4 t_6 t_1
212
201

Figure 2.5. An example that illustrates the situation where the Max-Min heuristic outperforms the Min-Min heuristic for an ETC matrix with high ATK and low ATS. Shown in the figure are the ETC matrix, the values of the statistical measures for the matrix, and a pictorial representation of the mappings produced by the Max-Min and the Min-Min heuristics.

the Max-Min heuristic outperforms Min-Min. A pictorial representation of the mapping produced by both heuristics is given in the figure. The makespan for the mapping produced by Max-Min is 212 and the makespan for the mapping produced by Min-Min is 243. Although ATS for the ETC matrix in this figure is 0, the ETC matrix still has the property that there are few tasks that have a high execution time compared to the rest of the other tasks.

An example where Min-Min outperforms Max-Min for an ETC matrix with low ATK is given in Figure 2.6. The ETC in the figure has a low ATK value of –0.72. A representation of the mappings produced by the two heuristics is given in the figure. The makespan of the mapping produced by Min-Min is 197 and the makespan of the mapping produced by Max-Min is 212.

2.4.3. Selection Based on One Measure

One way to make heuristic selections is to identify a single measure that has the most correlation with the performance of the studied heuristics, and then identify the ranges of values within which a heuristic performs better. This method is a simple method and the selection decisions based on it are straightforward. However, in some cases this method may lead to a high number of wrong decisions. Consider the example in Figure 2.7. This figure represents the performance of two heuristics *A* and *B* relative to the values of two heterogeneity measures 1 and 2. For selections based on a single measure, the best measure to use is measure 1 and the best decisions can be made if heuristic *A* is used when the value of measure 1 is between 0.5 and 1.0, and heuristic *B* is used otherwise. The number of wrong decisions in this case will be 24. The next section illustrates how a regression tree can be used to make better decisions based on

	m_1	m_2
t_1	62	44
t_2	42	55
t_3	49	45
t_4	48	52
t_5	55	56
t_6	40	51
t_7	54	52
t_8	52	62

AMC = 0.08
ATC = 0.12
AMS = 0
ATS = 0.08
AMK = − 2
ATK = − 0.72

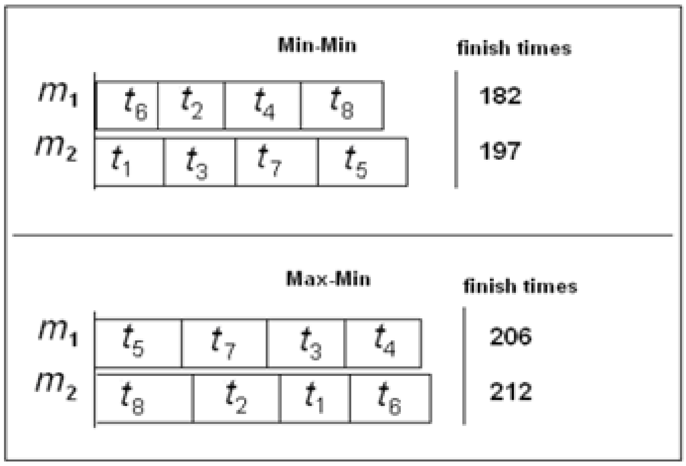


Figure 2.6. An example ETC matrix that illustrates the situation where the Min-Min heuristic outperforms the Max-Min heuristic for an ETC matrix with low kurtosis. Shown in the figure are the ETC matrix, the values of the statistical measures for the matrix, and a pictorial representation of the mappings produced by the Max-Min and the Min-Min heuristics.

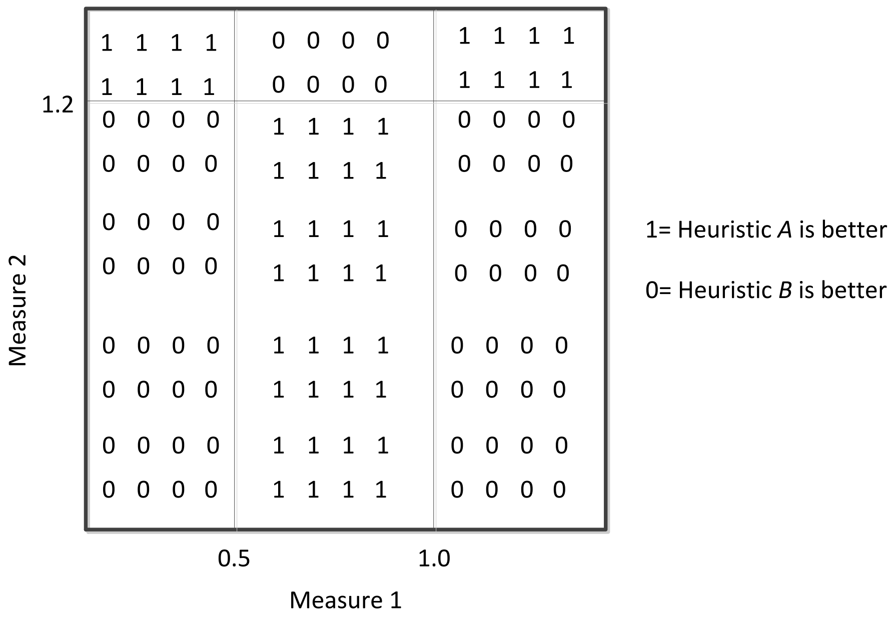


Figure 2.7. An example to illustrate heuristic selection decisions.

multiple measures, which will result in no wrong decisions if it is used to combine both measures 1 and 2 to make heuristic selection decisions for Figure 2.7.

2.4.4. Selection Based on Multiple Measures

A regression tree [BrF84] is a technique used to predict the outcome of a dependent variable based on a number of input variables. Regression trees are binary. The nodes of the tree represent yes/no questions about the input variables. If the answer to a question is yes, then the path along the left edge is followed, otherwise, the path along the right edge is followed. The question at each subsequent node is answered until a leaf node is reached. The leaf node represents a prediction of the value of the dependent variable based on the input variables. The prediction is simply the average of all of the values of the dependent variable that belong to that node.

Following the CART technique proposed in [BrF84], a regression tree is constructed by recursive partitioning of the data. At the beginning of the partitioning procedure, the question that will lead to the least error (quantified by the mean square error of the prediction) in the prediction is considered the root node of the tree. After the first partition, two sub-problems are created. Each sub-problem is solved in a similar manner as the original problem. A node is considered for partitioning if the following two conditions are true: 1) the number of values that belong to that node is greater than the minimum node size, and 2) there exists a split that leads to a decrease in the overall mean square error greater than or equal to the minimum decrease in error. Both minimum node size and minimum decrease in error are parameters set by the user. The minimum node size value that we have used is 500 and the minimum decrease in error value that we have used is 0.01

The regression tree for the example in Figure 2.7 is given in Figure 2.8. Measure 1 is at the root of the tree because it is the one that has the most correlation with the heuristics' performance, i.e., choosing the first split of based on the measure 1 results in the least error. This regression tree has 100% accuracy for the values in Figure 2.7.

To show how the performance of other heuristics is correlated with the statistical measures, we studied ETC matrices generated randomly and that have more tasks and machines than the matrices presented in this section. In the next section, we describe the result of the study.

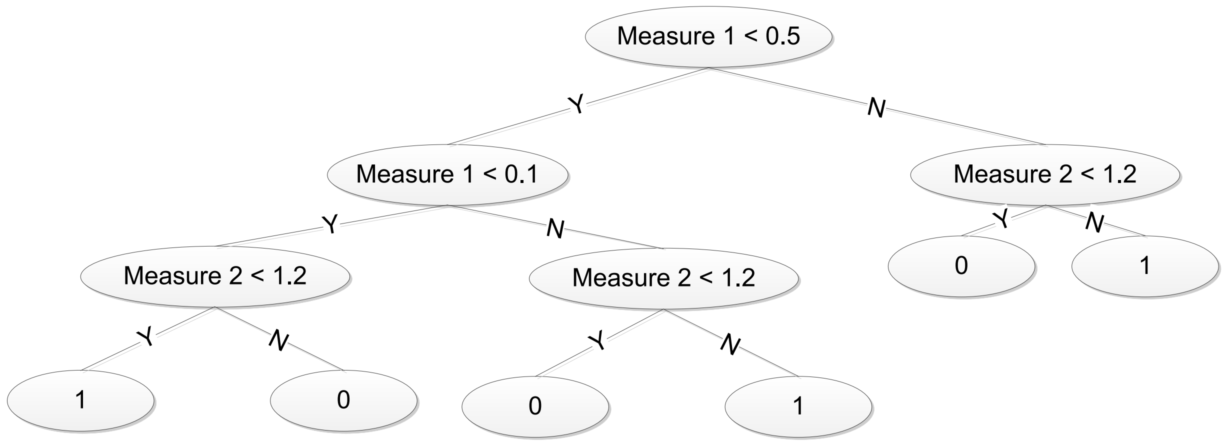


Figure 2.8. The regression tree that corresponds to the example given in Figure 2.7.

2.5. Synthetic Data

2.5.1. ETC Matrix Generation

Each of the ETC matrices that was used in this section was generated via the coefficient-of-variation-based method (CVB) proposed in [AlS00]. The CVB method uses the COV to represent task and machine heterogeneity. To generate an ETC matrix, the CVB method takes three parameters: (a) task COV, (b) machine COV, and (c) the mean task execution time.

The CVB method uses the Gamma distribution [Gub06] to generate the ETC values of an ETC matrix. The gamma distribution has a positive skewness. Furthermore, the COV, skewness, and kurtosis of the gamma distribution are correlated. Therefore, to show the effect of different combinations of COV, skewness, and kurtosis we have used the following seven distributions to generate the ETC values: 1) uniform, 2) gamma, 3) exponential, 4) Chi-square, 5) Cauchy, 6) normal, and 7) modified gamma distribution (described later in this section). Each of the distributions has different correlations between the COV, skewness, and kurtosis. The parameters of each of the distributions can be calculated based on the mean and COV values. The mean and COV values used to generate the ETC matrices are different for different simulations. The Cauchy distribution does not have a mean value. Therefore, the median was used as an estimate of the mean value. Any generated random value that is less than or equal to 0 is discarded.

The modified gamma distribution is obtained from the gamma distribution by truncating it at an upper limit value, normalizing it, and then inverting it. The truncation and normalization are done by discarding any generated values that have values higher than the upper limit. The inversion is done by subtracting the generated values from the upper limit value. Therefore, the modified gamma distribution will have negative skewness. The upper limit value equals the mean of the gamma distribution multiplied by an upper limit multiple u that is greater than 1. For our studies, we used *u* = 2. The procedure for generating the modified gamma distribution is depicted in Figure 2.9. Each ETC matrix used in the simulations has 128 tasks and eight machines.

After each ETC matrix was generated using a specific mean, task COV, and machine COV, we calculated the statistical measures of the generated ETC matrix to obtain their actual values. The actual average machine and average task COV values of the generated ETC may differ from those used to generate the ETC matrix due to a finite number of values being generated. The maximum value of the COV was selected to be 1.5, based on preliminary experiments that showed no significant changes in performance among the heuristics studied for ETCs with larger COVs.

2.5.2. Data Analysis: Scatterplots

The scatterplots shown in this section represent a way to make heuristic selection decisions based on one measure. Figure 2.10 shows the correlation between the relative makespan of Min-Min to Max-Min and the ATS. Because of the correlation between the COV

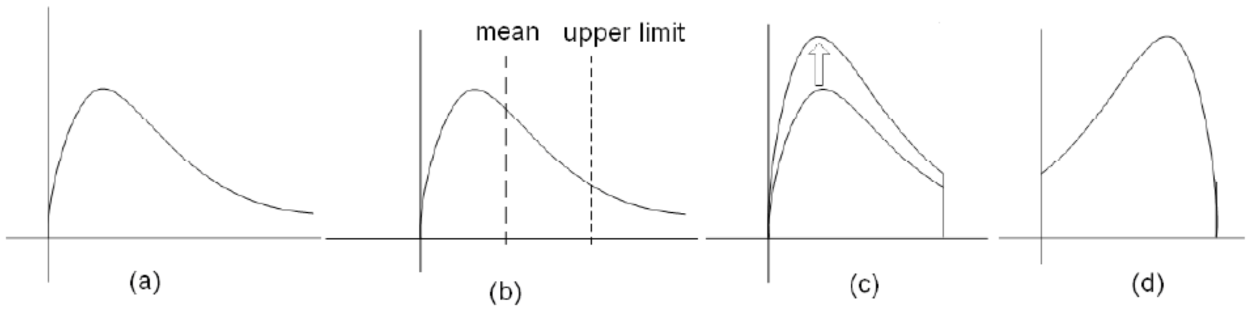


Figure 2.9. The procedure for generating the modified gamma distribution, (a) the gamma distribution, (b) the truncation of the gamma distribution at the upper limit value based on *u*=2 (i.e. the upper limit value will be twice the mean), (c) the normalization of the truncated distribution, and (d) the final modified gamma distribution (inverted).

and the skewness of the gamma distribution, we do not know if ATS affects the relative makespan, or if the effect of ATS is a result of it being correlated with ATC and only ATC having the effect on the relative makespan. Therefore, in Figure 2.10, we used both the modified gamma distribution and the gamma distribution to generate the ETC matrices. These two distributions have different correlations between their skewness and COV. Some of the ETC matrices (generated using the modified gamma distribution) have negative ATS values and have high ATC values; for those ETC matrices, Min-Min performs better than Max-Min. However, the ETC matrices (generated using the gamma distribution) that have high positive ATS values also have high ATC values; for those ETC matrices, Max-Min performs better. Therefore, we can see that there is a correlation between the ATS and the relative makespan of Min-Min to Max-Min.

For both distributions, the mean value was fixed at 20, the machine COV was fixed at 0.1, and the task COV was increased from 0.01 to 1.5. After the ETC matrices were generated, the actual ATS value was calculated for each ETC matrix. As shown in the figure, Max-Min outperforms Min-Min for ATS values greater than 1.4. However, for ATS values less than 0.5,

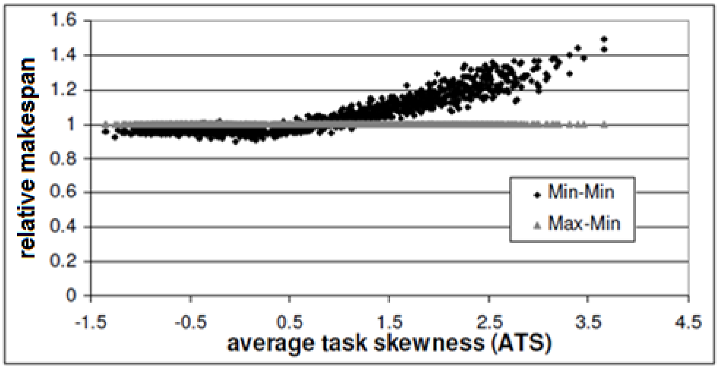


Figure 2.10. Scatterplot of the relative makespan of Min-Min to Max-Min for 1445 ETC matrices generated using both the gamma distribution and the modified gamma distribution. The CVB machine COV in this figure is fixed at 0.1 and the CVB task COV is increased from 0.01 to 1.5 for both distributions.

Min-Min always outperforms Max-Min. We do not see a decrease in the relative makespan of Min-Min for negative ATS ETC matrices. This is because ETC matrices with negative ATS have very few tasks that have low execution times and the majority of the tasks have execution time values close to the mean task execution time. Therefore, making better choices of assigning the few low execution time tasks will not lead to a large performance gain.

The effect of AMC on the relative makespan of Min-Min to Max-Min is shown in Figure 2.11. The gamma distribution was used to generate the ETC matrices in the figure. The mean ETC value was fixed at 20, the task COV was fixed at 0.7, and the machine COV was increased from 0.01 to 1.5. Min-Min almost always outperforms Max-Min for AMC values greater than 0.5. The reason Min-Min's performance relative to Max-Min's performance becomes better as the AMC value increases is because as the AMC increases the average difference between the best performing machine and the worst one becomes larger. Therefore, a task that has a lower execution time for a specific machine will have a higher ratio between the execution time on the best machine and the execution time on other machines. Assigning those tasks first (which Min-Min does) will result in a lower makespan.

The scatterplots in this section represent a simple way to compare the relative makespan and make heuristic selection decisions based on only one measure. In the next section, we show how regression trees can be used to compare the relative makespan of heuristics based on all of the six heterogeneity measures.

2.5.3. Data Analysis: Regression Trees

In our study, we have used regression trees to predict whether a heuristic is better than another for a specific ETC matrix based on the values of the heterogeneity measures for that

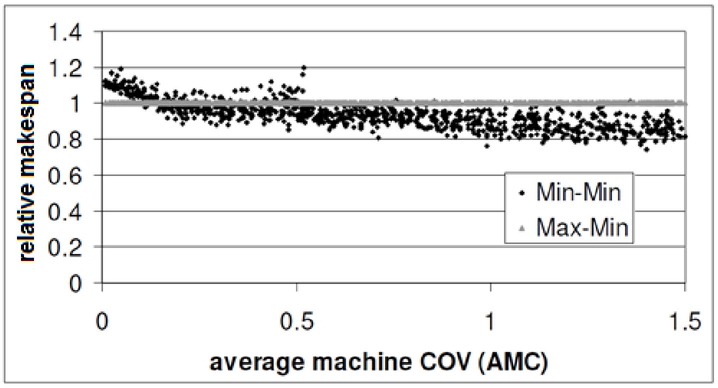


Figure 2.11. Scatterplot of the relative makespan of Min-Min to Max-Min for 925 ETC matrices generated using the gamma distribution. The task COV is fixed at 0.7, and the machine COV is increased from 0.01 to 1.5.

matrix. We have generated 100,000 ETC matrices randomly from the seven distributions described in Section 2.5.1. The mean ETC value used to generate the matrices is 500. The machine COV and the task COV values used to generate each matrix were sampled from a uniform distribution between the values of 0 and 1.5. For each matrix, the makespan of each of the five studied heuristics is calculated.

For any two heuristics *A* and *B*, let *b* be a variable that may have a value of either 0 or 1, such that if the makespan of *B* is less than *A*, then $b = 0$, otherwise, $b = 1$. Each regression tree in this section attempts to predict the value of *b* for two heuristics based on the heterogeneity values of the corresponding ETC matrices. The values at the leaf nodes of each tree represent the average of *b* for the ETC matrices that belong to that node. The closer the values are to 0 or 1 the better the prediction. For a given leaf node, if the predicted value of *b* is greater than 0.5, then we will consider using heuristic *A* for all ETC matrices that belong to that node, otherwise, we will use heuristic *B*.

Figure 2.12 shows the regression tree that attempts to predict the value of *b* when heuristic *A* is Max-Min and heuristic *B* is sufferage. The value in each of the leaf nodes represents the ratio of the number of times that Max-Min was better or equal to sufferage for all

ETC matrices that belong to that node. Therefore, the number of wrong predictions of the tree can be calculated as follows. First, for each of the leaf nodes that have a value less than 0.5, multiply the node value by the number of ETC matrices that belong to that node, then sum across these leaf nodes. This value represents the number of times that we have chosen sufferage when Max-Min had better or equal performance. Second, for each of the leaf nodes that have a value greater than 0.5, multiply (1 – node value) by the number of ETC matrices that belong to that node, and then sum across these leaf nodes. This value represents the number of times that we have chosen Max-Min when we should have chosen Sufferage. The total number of wrong predictions is the sum of the values obtained by the previous two steps. For the regression tree in Figure 2.12, the total number of wrong predictions is 1064.

Among all of the 100,000 ETC matrices, sufferage is better than Max-Min for 95,252 of them. Therefore, without any knowledge about the values of the heterogeneity measures, a reasonable prediction would be to always use sufferage which will lead to 4748 wrong predictions. However, if we use the tree in Figure 2.12, the number of wrong predictions can be reduced by 78%. In addition, for each of the leaf nodes 1, 2, and 3, choosing Max-Min rather than sufferage decreased the makespan. Node 3 has the highest average decrease in makespan for each ETC matrix that belongs to that node; the average decrease is 10.3%. Nodes 1 and 2 have less average decrease in the makespan. Note that the skewness measures did not have significant effect on the decision of which of the two heuristics to use. Therefore, they were not used to partition any node.

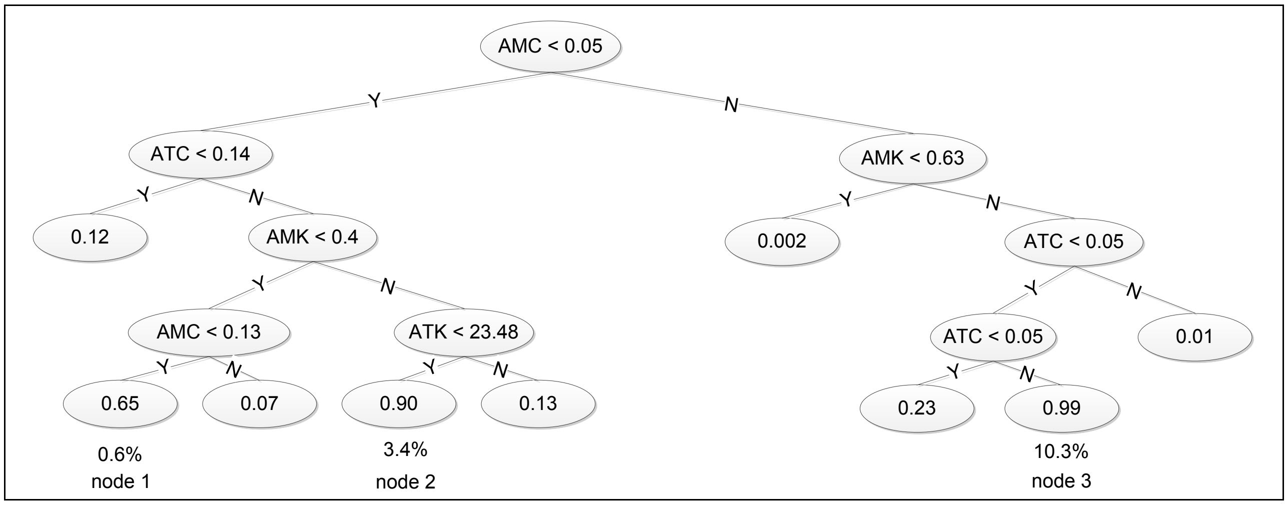


Figure 2.12. The regression tree for Max-Min vs. Sufferage. The percentages below some of the leaf nodes represent the average decrease in the makespan achieved by using Max-Min rather than Sufferage for each ETC matrix that belongs to that node. Each of the values in the nodes is rounded to two decimal places except when that results in a 0 value.

Figure 2.13 shows the regression tree for the same *b* used in Figure 2.12. However, in that tree, only the COV measures were used to make predictions. The number of wrong predictions in that tree is 2255. Therefore, using the kurtosis measures in Figure 2.12 reduced the number of wrong predictions by 53%.

We have also built a regression tree for predicting the value of *b* when heuristic *A* is Max-Min and heuristic *B* is MCT. This tree is given in Figure 2.14. For this tree, only three of the measures where used to make decisions, namely, AMC, ATC, and ATK. The regression tree when heuristic *A* is Max-Min and heuristic *B* is KPB is identical to the one in Figure 2.14. This is due to the similarity between the procedures of both heuristics.

It is important to note that the results shown in this section apply for the distributions used to generate the ETC matrices. Other HC systems that have ETC matrices that belong to different underlying distributions may have different regression trees. For such systems, the

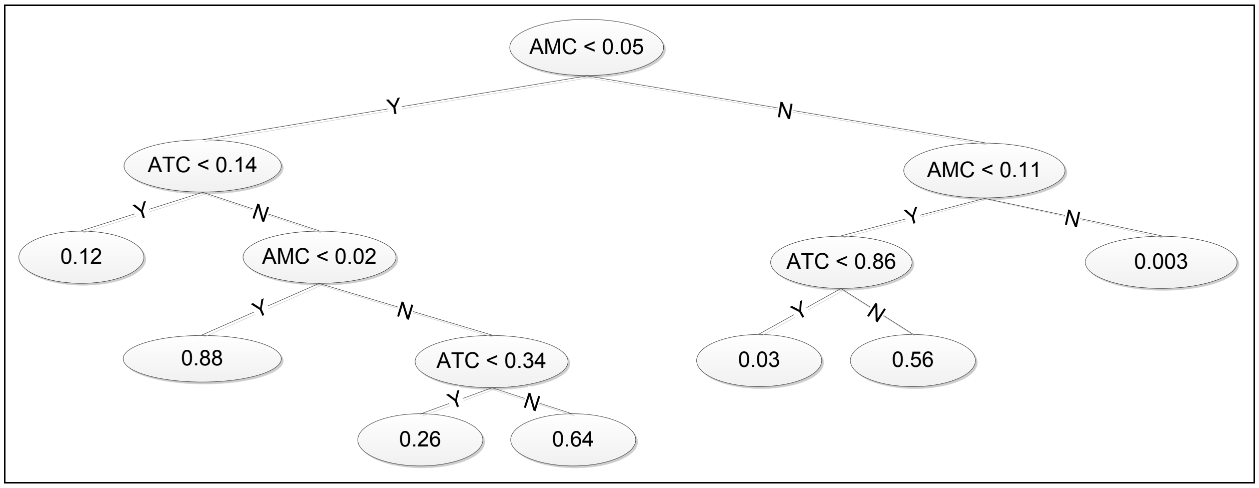


Figure 2.13. The regression tree for Max-Min vs. Sufferage when only the COV measures are used.

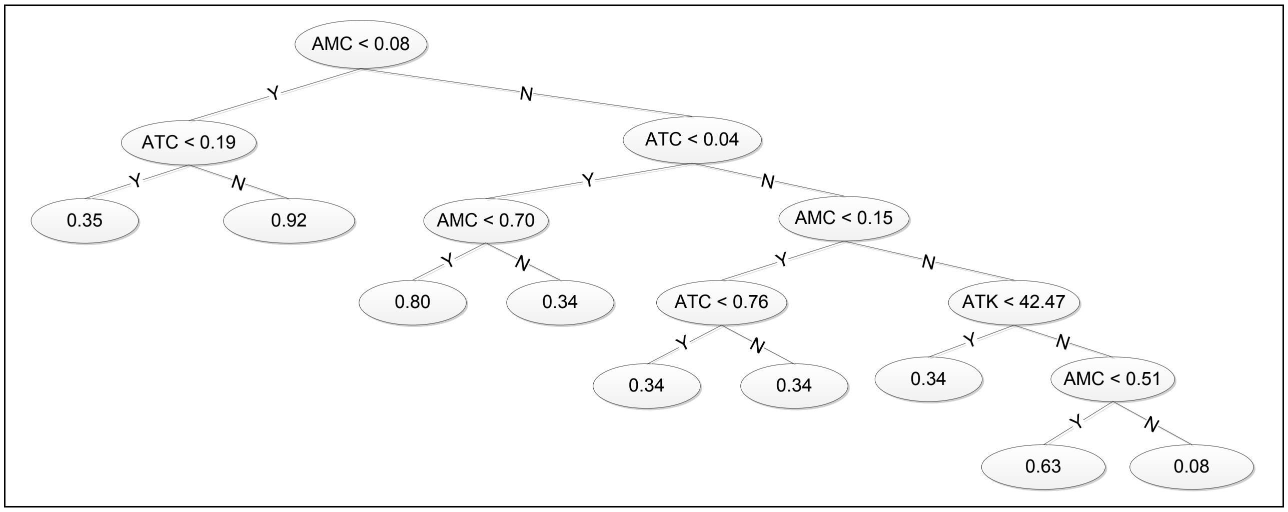


Figure 2.14. The regression tree for Max-Min vs. MCT.

average percentage decrease in the makespan when using a regression tree compared to using a heuristic *H* all the time can be computed as follows. First, multiply the average percentage decrease in makespan at each leaf node where *H* is not chosen by the number of ETC matrices that belong to that node. Then, take the average across all leaf nodes where *H* is not chosen. Finally, divide the result by the total number of ETC matrices used to construct the tree.

2.6. Related Work

ETC matrices were previously used with different degrees of heterogeneity (e.g., [AlB05, BaS01, BrS01, BrS08, MaA99, WuS00]). Most of these ETC matrices were generated by the range-based method described in [BrS01] and the CVB method described in [AlS00]. Therefore, depending on which method was used, heterogeneity was assumed to be either the range of the execution time values, or the COV.

Regression trees have been used widely in machine learning, decision making, pattern recognition, and data mining. For example, in [CoR10], a number of machine learning techniques including regression trees have been used to predict the performance of total order broadcast algorithms in systems running heterogeneous workloads. A number of workload and system characteristics were used as input variables for predicting the performance in terms of message delivery latency and throughput. Another example of using regression trees is given in [PoC08]. In that work, the authors propose the use of regression trees to predict the performance of transactional memory workloads based on different hardware transactional memory design dimensions and multicore microarchitecture configuration.

2.7. Conclusions

In this chapter, we proposed a number of statistical measures to quantify the heterogeneity of HC systems. A method to calculate each of the measures for an existing ETC matrix was described. The impact that the heterogeneity measures may have on the performance of five different heuristics was demonstrated through simple examples. In addition, ETC matrices generated randomly via seven different distributions have been used to show how using regression trees to analyze the information provided by the heterogeneity measures allows us to make better heuristic selection decisions.

Chapter 3

Characterizing Heterogeneous Environments using Singular Value Decomposition

3.1. Introduction

Many computing environments are heterogeneous, i.e., they consist of a number of different machines that vary in their computational capabilities. These machines are used to execute task types that vary in their computational requirements. Different task types can be better suited to different machine architectures. Further, while a machine *A* may be better than a machine *B* for one task type, it may not be better for another task type; performance is a function of the interaction of a machine's capabilities and a task type's requirements. We use the term task type to refer to an executable program than can be run many times. A task is an instance of a task type that is executed once.

It is common to arrange the estimated time to compute (ETC) of task types on machines in an ETC matrix. Entry (i, j) in the ETC matrix represents the ETC of task type i on machine j. The ETC values can be based on user supplied information, experimental data, or task profiling and analytical benchmarking (e.g., [AlB05, FrS93, GhY93, KhP93, MaB99, YaA93, YaK94]). The determination of ETC values is a separate research problem; the assumption of such ETC information is a common practice in resource allocation research (e.g., [BaS01, BrS11, DhA02, GhY93, KaA98, KhP93, LeP95, SiY96, XuN01]). An ETC value is the estimated time to

A preliminary version of portions of this chapter appeared in [AlM10, AlM11a].

compute a given task type on a given machine when it is run alone.

Quantifying the heterogeneity of a heterogeneous computing (HC) environment is important and has multiple useful applications. Examples of such applications include, predicting the performance of HC environments [ChM10], selecting appropriate heuristics to use in an HC environment based on its heterogeneity as shown in Chapter 2, "what-if studies" to identify the effect of adding/removing task types or machines from an HC system on its heterogeneity, and generating ETC matrices for simulation studies that span the entire range of heterogeneities [AlM10]. The purpose of this chapter is to provide heterogeneity measures that can be used as a standard way to compare different heterogeneous computing environments.

Although characterizing the heterogeneity of HC environments is important, there has not been much research in this area. In [AlS00, BrS01], methods for generating HC environments, based on ETC matrices, for simulation studies were proposed. The method in [AlS00] has been used widely, e.g., in [CaJ09, EsA09, HuY09, KhA06]. However, these methods do not deal with the problem of characterizing the heterogeneity of *existing* HC environments. To the best of our knowledge there is no other research that deals with the problem of identifying standard measures for quantifying the heterogeneity of computing environments.

There can be many methods to characterize the heterogeneity of an HC environment. In addition, the measured value of the heterogeneity of the environment may vary widely depending on the methods used. Therefore, we are motivated to determine standard measures of heterogeneity.

We have identified some properties that heterogeneity measures should have. These properties directed our choice of the heterogeneity measures. First, a heterogeneity measure should match intuition and common beliefs about heterogeneity. Second, it should not be affected by multiplying the ETC matrix by a scaling factor. This is because the ETC values can be represented in different time units (e.g., seconds vs. minutes). Third, when multiple measures are used to examine different aspects of heterogeneity, they should be as independent as possible of each other (i.e., we should be able to change the value of one of the measures independent of the others). There is no value of having two or more measures that are totally correlated. It would be sufficient to just use one of them. For example, if the standard deviation was used to represent the heterogeneity of a set of values, then there is no value of using the variance as another measure of heterogeneity because both measures will be totally correlated.

In this chapter, we introduce three measures for characterizing the heterogeneity of computing environments. These measures are: machine performance homogeneity (MPH), task easiness homogeneity (TEH), and task-machine affinity (TMA).

We have identified a computational procedure that puts a matrix, which represents an HC environment, in standard form. The standard form enables us to have the three independent heterogeneity measures: MPH, TEH, and TMA (satisfying the third property for heterogeneity measures). Putting the matrix in standard form also allows us to simplify the calculation of the TMA.

In summary, the contributions of this chapter are: a) to introduce three heterogeneity measures, b) to illustrate how the singular value decomposition (SVD) can be used to calculate TMA, c) to determine a standard matrix form that keeps the three measures independent and allows a simplified calculation of TMA, and d) to illustrate how the measures can be used to analyze some real world environments obtained from the SPEC benchmarks.

The rest of this chapter is organized as follows. Section 3.2 discusses the singular value decomposition. The proposed three heterogeneity measures are given in Section 3.3. Section 3.4

gives examples of heterogeneous environments to illustrate the motivation behind the measures that we have introduced in this chapter. Examples of HC environments that are based on real world data from the SPEC benchmarks are given in Section 3.5. Section 3.6 describes the special cases of HC environments for which the standard form matrix cannot be determined. Finally, conclusions are presented in Section 3.7.

3.2. Singular Value Decomposition

Singular value decomposition (SVD) is a standard matrix decomposition that can be performed on any arbitrary matrix [GoV89] that is used widely for determining the rank or condition number of a matrix, computing the best low-rank matrix approximation, and solving a host of pattern recognition problems for a wide range of applications. For an m-by-n real matrix A, there exists a factorization in the form:

$$A=U\Sigma V^T, \tag{3.1}$$

where U is an m-by-m orthogonal matrix, Σ is an m-by-n diagonal matrix with non-negative entries in descending order, V is an n-by-n orthogonal matrix, and ^{T} denotes the transpose. The diagonal values of the matrix Σ are the singular values, denoted σ_i, such that

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0.$$

The magnitudes of the singular values represent the degree of linear dependence among the columns (or rows) of a matrix. The number of singular values is equal to min(m,n). For our application, it is more common to have more task types than machines so that the number of singular values will be equal to n, the number of columns (machines). The SVD of a matrix with $m > n$ can be calculated in $O(mn^2)$ operations. Section 3.3.5 illustrates how the SVD is used to calculate the TMA. For the purposes of this chapter, $m = T$, is the number of task types and $n = M$, is the number of machines.

3.3. Heterogeneity Measures

3.3.1. Estimated Computation Speed Matrix

Another way of representing an HC environment is by using an estimated computation speed (ECS) matrix. The ECS matrix can be obtained from an ETC matrix by taking the reciprocal of each entry in the ETC matrix, i.e.,

$$\text{ECS}(i,j) = 1/\text{ETC}(i,j) \tag{3.2}$$

Entry (i, j) of the ECS matrix represents the amount of task type i that can be completed in a unit time on machine j. Therefore, larger entries in the ECS matrix correspond to more powerful machines for a specific task type.

In some HC environments, some machines may not be able to run specific task types because of specific task type requirements (e.g., specific architecture requirements or operating system requirements). In the ECS matrix, if task type i cannot run on machine j, then entry (i, j) will be equal to 0. The corresponding entry in the ETC matrix would be equal to ∞. Both the ETC and ECS matrices are non-negative matrices. Although individual entries in the ECS matrix can be equal to 0, there cannot be columns with all 0 entries or rows with all 0 entries because both cases correspond to a machine that cannot execute any task type or a task type that cannot be executed on any machine, respectively.

3.3.2. Machine Performance Homogeneity

One way to measure the performance of a machine is by the sum of the values along the corresponding column in the ECS matrix. For example, the performance of machine 1 for the

ECS matrix in Figure 3.1 is 17. Higher column sums correspond to machines with better performance for the given task types in the ECS matrix. The performance of machine j, $\text{MP}_{j,}$ for an ECS matrix with T task types is given by

$$\text{MP}_j = \textstyle\sum_{i=1}^{T} \text{ECS}(i,j)\ . \tag{3.3}$$

Clearly, if all the machines' performances are equal, then we have a completely homogeneous computing environment in terms of machine performance. However, when the performances are not equal, there can be a number of different ways to combine the performance values to measure machine performance homogeneity (or heterogeneity).

Let the machines of the ECS matrix be sorted in ascending order of their performance (i.e., the columns are ordered in ascending order of their sums). We define the machine performance homogeneity (MPH) measure to be equal to the average ratio of a machine performance to its next better performing machine, i.e., for an ECS matrix with M machines,

$$\text{MPH} = \frac{\sum_{j=1}^{M-1}\left(\text{MP}_j/\text{MP}_{j+1}\right)}{M-1}. \tag{3.4}$$

The MPH of the ECS matrix in Figure 3.1 is 0.52.

The weighting factor ($\underline{w_{t_i}}$) of task type i can be used to represent a number of characteristics (e.g., the importance of the task type, the number of times that a task type is executed, or the probability that a task type will be executed). Similarly, the weighting factor

	m_1	m_1	m_1
t_1	6	10	20
t_2	5	3	10
t_3	3	5	13
t_4	3	4	40

Figure 3.1. An example ECS matrix to illustrate how machine performance is calculated.

(w_{m_j}) of machine *j* can be used to represent a number of characteristics (e.g., security level of that machine).

The weighting factors are incorporated in the equations for calculating each of the measures presented in this chapter. These factors make the measures more flexible, which enables them to be applied to a wide variety of environments. Therefore, the formula for calculating machine *j*'s performance, when weighting factors are used, can be generalized to

$$\mathrm{MP}_j = w_{m_j} \sum_{i=1}^{T} w_{t_i} \mathrm{ECS}(i, j). \tag{3.5}$$

3.3.3. MPH Compared to Other Measures

We compare MPH with other possible measures and show that MPH has the first property of a heterogeneity measure (i.e., it matches the intuition about heterogeneity) while the other measures do not have that property. Other possible measures include: 1) the ratio, *R*, of the performance of the lowest performance machine to that of the highest performance one, as a measure of homogeneity (i.e., higher values correspond to more homogeneous environments), 2) the geometric mean[1], *G,* of the ratios of the performance of the lower performance machine to that of the higher performance machine in each pair of adjacent machines in the ECS matrix, as a measure of homogeneity, and 3) the coefficient of variation[2], COV, of the machines' performances, as a measure of heterogeneity. All of the above measures have the second property of a heterogeneity measure (i.e., they are not affected by scaling the performances by a common factor).

[1] The geometric mean of a set of *n* values a_i is given by $(\prod_{i=1}^{n} a_i)^{1/n}$.
[2] The COV of a set of a set of *n* values a_i with standard deviation *S* and mean μ is given by *S*/ μ.

Figure 3.2 shows four examples of possible machines' performances of HC environments with five machines. Intuitively, Environment 1 is the most heterogeneous because none of the machines performances are equal. Environments 2 and 3 have the same heterogeneity because they both have four machines with the same performance and the ratio of the performance of the most powerful machine and the performance of the least powerful one is 1/16. Environment 4 has three machines that have the same performance. Therefore, it is less heterogeneous than environment 1 and more heterogeneous that environments 2 and 3. In the figure, the values of each of the measures for each of the environments are given. The only measure that matches intuition is the MPH measure. Both measures, *G* and *R*, capture the heterogeneity between the highest performance machine and the lowest performance machine. However, they do not capture the spread of the performance of the intermediate machines.

3.3.4. Task Easiness Homogeneity

The easiness of a task type is quantified by the sum of the ECS values of that task type over all machines (i.e., the corresponding row sum in the ECS matrix). Task types with higher row sums are considered easier (i.e., they can be computed faster). Because the easiness of task

Environment 1.
1, 2, 4, 8, 16
MPH = 0.5, *R* = 0.06, *G* = 0.5, COV = 0.88
Environment 2.
1, 1, 1, 1, 16
MPH = 0.77, *R* = 0.06, *G* = 0.5, COV = 1.5
Environment 3.
1,16, 16, 16, 16
MPH = 0.77, *R* = 0.06, *G* = 0.5, COV = 0.46
Environment 4.
1, 4, 4, 4, 16
MPH = 0.63, *R* = 0.06, *G* = 0.5, COV = 0.90

Figure 3.2. Machines' performances of example HC environments.

types can vary widely, a measure of task type easiness is needed. In this section, we present a measure for task type easiness homogeneity (TEH) and show how it is calculated.

For the canonical ECS matrix, let both the machine performances and task easiness values be sorted in ascending order. Formally, the canonical form ECS matrix is a matrix where

1. $\mathrm{MP}_j \leq \mathrm{MP}_{j+1}$ for $0 < j < M$, and
2. $\mathrm{TE}_i \leq \mathrm{TE}_{i+1}$ for $0 < i < T$.

The calculation procedure for TEH is similar to that of MPH. However, the homogeneity is calculated for task types (rows). Let $\underline{\mathrm{TE}_i}$ be the easiness of task type i. The general formula for calculating task type easiness, when weighting factors are used, is

$$\mathrm{TE}_i = w_{t_i} \sum_{j=1}^{M} w_{m_j} \mathrm{ECS}(i, j). \tag{3.6}$$

Following the same intuition behind MPH, we use a TEH measure that is equal to the average ratio of a task type's easiness to its next task type's easiness. The general TEH formula for a canonical ECS matrix is given by

$$\mathrm{TEH} = \frac{\sum_{i=1}^{T-1} (\mathrm{TE}_i / \mathrm{TE}_{i+1})}{T-1}. \tag{3.7}$$

Both, MPH and TEH take values in the interval (0, 1].

3.3.5. Task-Machine Affinity

MPH and TEH represent one aspect of heterogeneity; however, it they do not capture the case where various sets of task types are better suited to run on different sets of machines (i.e., task-machine affinity). For example, the two ECS matrices in Figure 3.3 are completely homogeneous in terms of machine performance and task easiness. However, the ECS in Figure 3.3 (b) is heterogeneous in the sense that some of the machines are better suited to execute different sets of task types. Therefore, in order to represent this different aspect of heterogeneity we introduce the TMA measure.

Let a **standard ECS** matrix be an ECS matrix with equal row sums and equal column sums. In the next subsection, we describe a procedure to obtain the standard ECS matrix. The standard ECS matrix is both task type and machine homogeneous, which enables us to calculate TMA independent of MPH and TEH (i.e., we satisfy the third property of a heterogeneity measure).

For a standard ECS matrix with T task types and M machines, there are min(T,M) singular values. For a given ECS matrix with equal row sums and equal column sums, a lower column (or row) correlation will correspond to larger values of the non-maximum singular values relative to σ_1 and an intuitively higher value of TMA. Therefore, we use the following formula to calculate TMA:

$$\text{TMA}=\left(\frac{\sum_{i=2}^{\min(T,M)}\sigma_i}{(\min(T,M)-1)}\right)/\sigma_1 \,. \tag{3.8}$$

	m_1	m_2	m_3
t_1	10	10	10
t_2	10	10	10
t_3	10	10	10

(a)

	m_1	m_2	m_3
t_1	5	10	20
t_2	20	5	10
t_3	10	20	5

(b)

Figure 3.3. Two example ECS matrices to demonstrate task-machine affinity: (a) an ECS matrix where there is no affinity between task types and machines, and (b) an ECS matrix that has a high degree of affinity between task types and machines.

3.3.6. The Standard ECS Matrix

To make the TMA measure independent of MPH and TEH, we compute the singular values from a standard ECS matrix where all the column sums are equal, and all the row sums are equal (i.e., the standard matrix is both machine and task type homogeneous). In this section, we illustrate how such a standard ECS matrix can be computed from an ECS matrix with all positive elements, which we will refer to as a positive matrix.

The problem of row and column normalization has appeared in other applications. For example, the requirement for row and column sum normalization occurs in the practical problem of estimating doubly stochastic matrices for certain types of Markov random processes. Motivated by this problem, Sinkhorn [Sin64] proved that for any positive square matrix A, there are two diagonal matrices D_1 and D_2 such that D_1AD_2 is doubly stochastic. In other words, given any positive square matrix, one can suitably scale the individual rows and columns in such a way that each row and column sums to the same value. Furthermore, the diagonal matrices D_1 and D_2 are unique up to scalar multiplication.

While Sinkhorn's theorem and its proof specifically apply to positive square matrices, the results can also, after suitable modifications, be used to convert a positive rectangular matrix through a series of row and column normalizations into a positive matrix with the property that the row sums are equal and the column sums are equal. This is illustrated in Appendix A. Hence, we have the following result.

Theorem 1. *For a $T \times M$ ECS matrix with positive elements, there are diagonal matrices D_1 and D_2 such that, for any nonzero scalar k, $D_1(ECS)D_2$ is a positive matrix whose rows each sum to Mk and whose columns each sum to Tk. Furthermore, D_1 and D_2 are unique up to scalar multiples.*

Sinkhorn provided an iterative procedure to obtain the required diagonal matrices and proved that the procedure converged. A similar iterative procedure is also applied in this work.

3.3.7. Simplified TMA Calculation

The singular values of the standard ECS matrix are related to the column sums and row sums. The following theorem shows that for a specific choice of the row sums and the column sums the maximum singular value of the standard ECS matrix will always be 1.

Theorem 2. *For a $T \times M$ ECS matrix with the property that each row sums to $\sqrt{M/T}$ and each column sums to $\sqrt{T/M}$, the largest singular value is equal to 1.*

The proof of Theorem 2 is given in Appendix B. If we let k (in Theorem 1) be equal to $\sqrt{1/MT}$, then it follows that we can convert an ECS matrix to a standard one that has the column sums equal to $\sqrt{T/M}$ and the row sums equal to $\sqrt{M/T}$. The maximum singular value of that standard ECS matrix will be equal to 1. This enables us to rewrite the equation for TMA (Equation 3.8) in the following simpler form

$$\text{TMA}=\sum_{i=2}^{\min(T,M)} \sigma_i/(\min(T,M) - 1)\,. \tag{3.9}$$

We use an iterative procedure to normalize the ECS matrix. The iterative approach generates a series of ECS_k matrices. The procedure switches between row normalization and column normalization until it converges to a row and column normalized matrix. The matrix ECS_k is defined by

$$\text{ECS}_k(i,j) = \begin{cases} \frac{\sqrt{T/M}\text{ECS}_{k-1}(i,j)}{\sum_{q=1}^{T} \text{ECS}_{k-1}(q,j)}, k = 1,3,5,\dots \\ \frac{\sqrt{M/T}\text{ECS}_{k-1}(i,j)}{\sum_{p=1}^{M} \text{ECS}_{k-1}(i,p)}, k =2,4,6,\dots \end{cases}. \tag{3.10}$$

In cases where the ECS matrix contains zero-valued elements, the iteration defined by the previous equation may not converge. This will be discussed further in Section 3.63.6.

3.4. Contrived Illustrative Examples

To further illustrate the intuition behind MPH, TEH, and TMA, we show, in Figure 3.4, some simple 2×2 ECS matrices and where they fall within the range of all possible values of the three measures. The examples have near extremal values for each of the measures. Entries with 0 values in the ECS matrix represent task types that cannot be executed on specific machines.

Intuitively, matrices *A*, *B*, *C* and *D* all have a very high task-machine affinity because there is at least one task type that can only be executed on one machine (i.e., that task type has the highest affinity to the corresponding machine). The TMA measure reflects this intuition. All these matrices have a TMA value of 1. Matrix *C* is already a standard matrix. The second singular value of that matrix is 1. When the procedure in Equation 3.10 is applied to matrices *A*, *B*, and *D* they all converge to the standard form of *C*. In contrast, matrices *E*, *F*, *G*, and *H* all have no task-machine affinity because the performance ratio of machines 1 and 2 is the same for both task types. The TMA value for these matrices is 0.

Matrices *C*, *D, G*, and *H* are all nearly homogeneous in terms of machine performance. The performances of both machines are nearly homogeneous over both tasks. All the matrices have high MPH values. In contrast, matrices *A*, *B*, *E*, and *F* are all very heterogeneous in terms of machine performance. For all four matrices, the performance of machine 2 is much better than machine 1. These matrices have low MPH values.

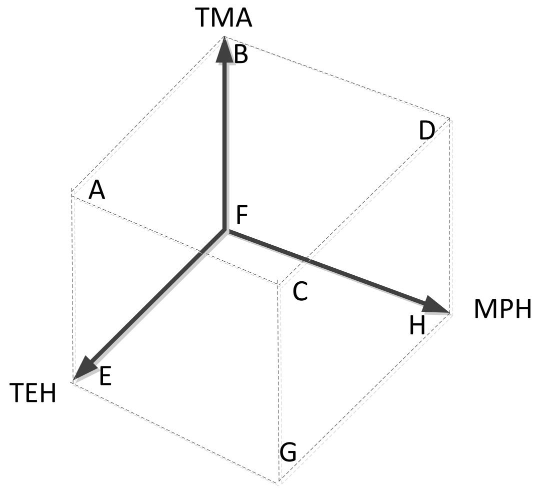
Matrices *A*, *C*, *E*, and *G* are all nearly homogeneous in terms of task type easiness. They have high TEH values. In contrast, the task types of matrices *B*, *D*, *F*, and *H* are all very heterogeneous. For all four matrices, task type 1 is much more difficult than task type 2. The four matrices have low TEH values.

3.5. Example ECS Matrices from SPEC Benchmarks

In this section, we show examples of ETC matrices extracted from the integer and floating point SPEC benchmarks (SPEC CINT2006Rate and SPEC CFP2006Rate) [Spe06]. The matrices illustrate how environments constructed from real world task types and machines can have widely varying values for each of the measures proposed in this chapter. Note that the benchmarks were used for illustration purposes only and the measures proposed in this chapter can be applied to any HC environment that is represented by an ETC matrix.

The SPEC CINT2006Rate consists of twelve task types. The SPEC CFP2006Rate consists of 17 task types. We have extracted the peak runtime values for five different machines. The machines have processors that have different architectures and are produced by different manufacturers. Figure 3.5 shows the five different machines. Figures 3.6 and 3.7 show the peak runtime values of each of the five machines for the SPEC CINT2006Rate and SPEC CFP2006Rate benchmarks, respectively.

Figure 3.8 shows two example 2×2 ETC matrices extracted from the values in Figures 3.6 and 3.7. The figures also show the values for each of the measures. The two matrices are almost identical in terms of machine performance homogeneity. However, the task type difficulty and task-machine affinity of the two matrices vary. The task types of matrix (a) are more homogeneous than the ones of matrix (b). Because the ratios of the performances of the two machines of matrix (b) vary widely for each task type, the TMA value for that matrix is



Coordinates (MPH, TEH, TMA)

A(0.05,1, 0.98)

0	100
1	100

B(0.01,0.01,1)

1	0
0	100

C(1,1,1)

1	0
0	1

D (1,0.05, 0.98)

0	1
100	100

E (0.01,1,0)

1	100
1	100

F(0.01,0.01, 0)

1	100
100	10000

G (1,1,0)

1	1
1	1

H(1,0.01,0)

1	1
100	100

Figure 3.4 Example extreme 2 × 2 ECS matrices with extremal values of each of the three measures: MPH, TEH, and TMA.

high. In contrast, the ratios of the performances of the two machines are very close for each of the task types of matrix (a).

We also have calculated the values of the three measures for the entire CINT matrix and the entire CFP matrix. The values are shown in Figures 3.6 and 3.7. The machine performance homogeneity and the task type easiness of both matrices are almost identical. However, for the floating point applications in the CFP matrix, task types have more affinity to machines than that of the integer applications in the CINT matrix.

The iterative normalization procedure for the CFP and CINT matrices converged in 7 and 6 iterations, respectively. Every iteration consists of one column normalization followed by one row normalization. The procedure stops when the maximum error in any column or row norm is less than $1/10^8$.

The benchmarks presented in this section are for general purpose CPUs. We expect HC environments that consist of special purpose computing resources (e.g., accelerators and GPGPUs) and tasks that are better suited to run on these resources to have higher TMA values and lower TEH and MPH values.

3.6. Issues with Standard Form Matrices

In the ECS matrix, it may be desirable to have entries equal to 0 that correspond to machines that cannot execute specific task types. However, when the ECS matrix has some entries that are equal to 0, the iterative normalization procedure described in Section 3.3.7 is not guaranteed to converge to a standard ECS matrix. Consider the following ECS matrix

$$\begin{pmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{3.11}$$

m_1 = ASUS TS100-E6 (P7F-X) server system (Intel Xeon X3470)
m_2 = Fujitsu SPARC Enterprise M3000
m_3 = CELSIUS W280 Intel Core i7-870
m_4 = ProLiant SL165z G7 (2.2 GHz AMD Opteron 6174)
m_5 = IBM Power 750 Express (3.55 GHz, 32 core, SLES)

Figure 3.5. The five machines used from the SPEC benchmarks.

	m_1	m_2	m_3	m_4	m_5
400.perlbench	625	974	627	563	1159
401.bzip2	971	1592	988	956	1363
403.gcc	642	1166	631	626	1026
429.mcf	267	946	266	463	416
445.gobmk	639	1124	642	719	957
456.hmmer	300	1698	205	391	920
458.sjeng	768	1756	779	912	1235
462.libquantum	537	686	548	431	523
464.h264ref	1082	1814	1092	1095	1804
471.omnetpp	557	1450	569	698	1850
473.astar	707	1235	714	699	1179
483.xalancbmk	414	758	415	414	793

TEH = 0.90   MPH = 0.82   TMA = 0.07

Figure 3.6. SPEC CINT2006Rate Data.

In its current form, this matrix is not normalized because the second row and third column sums are both 2 while the other row and column sums are 1. It can be shown that there exists no combination of row and column normalizations to convert this matrix to a standard ECS matrix. To see this, observe that although row and column normalizations affect the values of the individual elements of the ECS matrix, zero-valued elements remain zero-valued elements

	m_1	m_2	m_3	m_4	m_5
410.bwaves	677	1776	664	1238	528
416.gamess	777	2701	778	1267	2522
433.milc	845	2210	851	1056	1104
434.zeusmp	694	1132	706	741	1034
435.gromacs	576	838	592	481	1061
436.cactusADM	859	1204	845	105	345
437.leslie3d	704	1507	711	1207	499
444.namd	657	979	678	648	789
447.dealII	589	893	596	518	764
450.soplex	509	1635	520	1018	441
453.povray	253	465	264	319	451
454.calculix	531	954	551	471	1227
459.GemsFDTD	855	2431	874	1420	1588
465.tonto	693	1122	724	626	1146
470.lbm	1102	1348	1150	892	940
481.wrf	964	1349	939	858	1621
482.sphinx3	1694	2887	1668	1298	3148

TEH = 0.91 MPH = 0.83 TMA = 0.13.

Figure 3.7. SPEC CFP2006Rate Data.

	m_4	m_5
471.omnetpp	698	1850
436.cactusADM	105	345

TEH=0.16
MPH=0.31
TMA=0.05

(a)

	m_1	m_4
436.cactusADM	859	105
450.soplex	509	1018

TEH=0.28
MPH=0.30
TMA=0.60

(b)

Figure 3.8. Example 2 × 2 ETC matrices extracted form the SPEC CINT2006Rate and CFP2006Rate benchmarks. The values for each of the measures are given for both matrices.

and non-zero elements remain non-zero elements for any combination of row and column normalization. Consequently, those elements that are equal to 0 will remain 0 for any combinations of row and column normalizations. Therefore, if there was a normalized version of this ECS matrix, each of the four nonzero elements must equal 1 so that the first and last row sums are 1 and the first two column sums are 1. However, this results in the original matrix, which is clearly not normalized due to the fact that, unlike the other row and column sums, the second row and third column sums are not equal to 1.

The authors of [MaO68] have presented a sufficient condition for a matrix to be row and column normalized. We summarize their results here.

A square matrix A with non-negative elements is said to be decomposable if there are permutation matrices P and Q such that PAQ has the block form

$$\begin{pmatrix} \mathrm{A}_{11} & 0 \\ \mathrm{A}_{21} & \mathrm{A}_{22} \end{pmatrix} \tag{3.12}$$

where A_{11} and A_{22} are square matrices. In other words, a matrix A with non-negative elements is decomposable if one can reorder the rows and columns so as to obtain the above form. If one cannot do this, then the matrix is said to be fully indecomposable. For any square matrix A with non-negative elements that is fully indecomposable, there exists diagonal matrices D_1 and D_2 such that D_1AD_2 has equal row sums and equal column sums. An example of a simple matrix that fails the above condition is the matrix in Equation 3.11. This matrix is decomposable because it can be placed in the block form of Equation 3.12 by moving the last column to the front to obtain

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix}. \tag{3.13}$$

The upper left 1×1 matrix is A_{11} and the lower right 2×2 matrix is A_{22} matrix.

Note that while being indecomposable is a sufficient condition for being able to row and column normalize a positive matrix, it is not a necessary condition. This is illustrated by a diagonal matrix with positive elements on its diagonal. Such matrices are already in the decomposable form given in Equation 3.12, but they can be easily converted into the identity matrix through row and column normalization to obtain the desired form.

The case for rectangular matrices with non-negative components is handled in a similar way. In this case, consider an $m \times n$ matrix A with $m < n$ (for the case when $m > n$, one can transpose, do the row and column normalizations, and transpose back). In this case, one says that A is fully indecomposable if every $m \times m$ sub-matrix of A is fully indecomposable. In future work, we will investigate evaluating the TMA for ECS matrices that cannot be row and column normalized.

3.7. Conclusions

In this chapter, we introduced three heterogeneity measures: machine performance homogeneity, task easiness homogeneity, and task-machine affinity. We identified three properties that heterogeneity measures should have and show how our three measures possess these properties. The intuition behind each of the measures is shown through many illustrative examples and ones extracted from the SPEC benchmarks. We have also explained the importance of keeping the three measures independent. We illustrated how an iterative row and column normalization procedure can be used to keep the measures independent.

Chapter 4

Power and Thermal-Aware Workload Allocation in Heterogeneous Data Centers

4.1. Introduction

Over the last decade, the power consumption of data centers has been increasing at a rapid rate. In a report by the EPA [Epa07], it is estimated that the power consumed by servers and data centers has more than doubled between the years 2000 and 2006. In 2006, it is estimated that the power consumed by servers and data centers was 61 billion kWh, which is equal to 1.5% of the total U.S. electricity consumption that year. This amounts to $4.5 billion in annual electricity costs, equivalent to the power consumption costs of 5.8 million average U.S. households. Motivated by the need to reduce the power consumption of data centers, many researchers have been investigating methods to increase the energy efficiency in computing at different levels: chip, server, rack, and data center (e.g., [AbV10, ApY11, AyM01, BeA12, ChG10, GaC11, JaM11, KiS08, LiH06, MoC05, PaT10, ToW09, YoA11]).

In some cases, there are physical limitations on the amount of power available for data centers. For example, Morgan Stanley is no longer able physically to get the power needed to run a new data center in Manhattan [BrR10]. In a survey of data centers [FiH08], 31% identify power availability as a key factor limiting server deployment. The EPA report also indicates that about 50% of the power consumed in data centers is due to the infrastructure for power delivery and cooling. Therefore, minimizing the power consumed by the cooling infrastructure, can lead

A preliminary version of portions of this chapter appeared in [AlP12].

to significant overall power savings.

This chapter studies two problems. In Problem 1, we maximize the performance of a data center that is subject to both a total power consumption constraint ($\underline{P_{\mathrm{const}}}$) and thermal constraints. The power consumption of the data center is the sum of the power consumption of both Computer Room Air Conditioning (CRAC) units and compute nodes. We quantify the performance of the data center as the total reward collected from completing tasks in a workload by their individual deadlines. We consider a data center in the steady state where task flow rates, temperatures at compute nodes and CRAC units, and the power consumption of compute nodes become constant. Therefore, the performance is equivalently quantified by the total rate at which reward is collected (the total reward rate). In Problem 2, we minimize the power consumption of a data center while guaranteeing that the total reward rate does not drop below a specified threshold ($\underline{R_{const}}$[1]).

Performance states (P-states) of cores provide a trade-off between the power consumed by a core and its performance [Hew10]. *Lower P-states consume more power and provide better performance.* The relationship between the performance and power consumption of the P-states is non-linear. In most cases, the lowest P-state (P0) is not the P-state with the best trade-off between performance and power consumption [LoS01, XiL08].

P-state assignments in data centers are mainly done at the compute node level. In cases where the workload fluctuates, the P-state of one or more cores is increased when the node's utilization drops below a specified threshold (e.g., [ElK02, PaS06, ToW08]). However, in a power or performance constrained data center where the workload assigned to a core is constant, the utilization of each core (that is not turned off) in a specific P-state will be close to 100% to

[1] Appendix F provides a list of notations used in this chapter

avoid idle time. This is because we want to execute as many tasks as possible to obtain the maximum performance for a given power consumption. In this chapter, we show that our technique of assigning the P-states when considering the whole data center increases the overall performance (in terms of increased reward or reduced power consumption) of the data center.

The power consumed by compute nodes in the data center is dissipated as heat that is removed by the CRAC units. Our approach of assigning tasks and P-states to cores is thermal aware as it considers the temperature evolution effects of P-state assignments, which in turn affects the power consumed by the CRAC units. For both problems studied in this chapter, we show how each assignment can be expressed as an exact optimization problem. The P-state assignment part of the problem introduces integer constraints. The integer constraints make each assignment problem not scalable with respect to the number of cores in the data center. A simple relaxation of the integer constraints may introduce additional binary constraints that make each assignment problem not scalable. To address this, we propose novel and scalable assignment techniques for both problems. Each technique involves solving a set of scalable optimization problems. Our techniques are compared against a technique that only considers putting a core in the lowest P-state or turning off the core. We show that using our techniques results in notable performance improvements.

In summary, we make the following novel contributions. Our first contribution is that we model data centers that are either power constrained or reward constrained. The second contribution is that we express each assignment problem at the data center level as an optimization problem that includes P-state assignment. The decisions of this optimization problem are: the P-states of cores, the desired number of tasks per unit time allocated to a core, and the outlet CRAC temperatures. The known solution techniques to both optimization problems are not scalable due to integer constraints imposed by the P-states. The third contribution is a scalable assignment technique to find near-optimal solutions for each problem. The fourth contribution is a dynamic scheduler that assigns tasks to cores such that the actual rate of task execution is as close as possible to the desired rate. Finally, the fifth contribution of this chapter is showing, through simulations, the performance gains of applying our techniques to solve Problems 1 and 2 as opposed to current techniques.

The rest of this chapter is organized as follows. Section 4.2 discusses related work. The data center model is described in Section 4.3. In Section 4.4, the thermal constraints in the data center are described. The assignment problems and our solution to the problems are given in Section 4.5. Section 4.6 describes the simulation set up. Simulation results are shown in Section 4.7. Future work is given in Section 4.8. Section 4.9 gives the conclusions.

4.2. Related Work

In [ToW09], a control system for minimizing the power consumption in blade server enclosures is proposed. The power consumption of the blade server is minimized using three techniques: blade server consolidation, adjusting the speeds of the fans, and assigning P-states to processors. The P-state assignment is based on a simple utilization-based technique. A processor is assigned a P-state so that the utilization is never higher than 80%. However, as discussed in the introduction, this technique is not effective in a power or performance constrained data center because the utilization of each core should be close to 100%. The other two techniques (blade server consolidation and adjusting the speeds of the fans) can be used in future work in combination with our assignment technique to reduce the power consumption.

In [PaT10], it is shown that using an integrated approach to managing the cooling and computational resources in a data center is more efficient than if the two resources were managed independently. Their technique is similar to ours in that it trades-off power consumption with reward (from QoS). They trade-off power by deciding the amount of compute resources will be turned on at a compute node. In this chapter, we extend that work in two directions. First, we consider a power constrained data center and a reward constrained data center. Second, we show how assigning P-states at the data center level results in improved performance.

The P-state assignment problem for optimizing some objective in a computer system has been studied widely (e.g., [ApY11, AyM01, CaR05, XiL08, YoA11, YuV08]). The primary difference between our work and these studies is that our work considers the power consumed by the CRAC units in addition to the power consumed by compute nodes.

The thermal-aware scheduling problem has been previously researched (e.g., [AbV10, ChG10, MoC05, PaG10]). However, unlike our study, none of these papers consider P-state assignment.

Many other techniques to increase the energy efficiency of data centers exist. For example, the Open Compute Project started by Facebook proposes the following two techniques: 1) using a 480V electrical distribution system to reduce energy loss and 2) reusing hot aisle air in the winter to heat offices. Another example proposed by the Sustainable Ecosystems Research Group at HP is to use water evaporation for cooling instead of using compressors. Many of these techniques can be used in conjunction with our technique to obtain further performance gains.

4.3. Data Center Model

4.3.1. Overview

In this section, we give a detailed description of the workload and the different components of the data center. Data centers are typically arranged in a hot-aisle/cold-aisle configuration, as depicted in Figure 4.1. CRAC units draw hot air from the top and deliver cold air through the perforated floor tiles in the cold aisles. Compute nodes draw air from the cold aisles. The power consumption of compute nodes causes the temperature inside of the compute nodes to rise. The hot air inside a compute node is expelled into the hot aisle. Due to complex air flow patterns in a data center, some of the hot air exiting a compute node re-circulates into another compute node.

4.3.2. Workload

We assume that we have a set of $\underline{T}$ known task types. The arrival rate of tasks of type i is given by $\underline{\lambda_i}$. A reward $\underline{r_i}$ is collected for completing a task of type i by the task's individual deadline. The deadline of a task of type i is given by: deadline = arrival time + $\underline{d_i}$. The value of d_i would be specified by the user. In addition, we assume tasks can be dropped if their deadlines cannot be satisfied. The goal of Problem 1 is to maximize the total reward that is collected given a constraint on the total power consumption of the data center (the power consumption of compute nodes and CRAC units). The goal of Problem 2 is to minimize the total power consumption given a constraint on the total reward collected.

4.3.3. Compute Nodes

Let the number of compute nodes in the data center be $\underline{\mathrm{NCN}}$. Each compute node has a

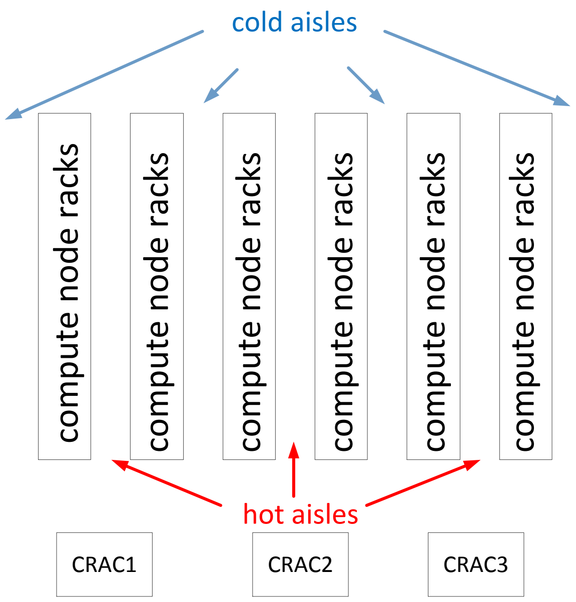


Figure 4.1. A hot-aisle/cold-aisle data center layout.

number of identical cores. Each compute node *j* belongs to a specific compute node type $\underline{\mathrm{NT}_j}$. Compute nodes with the same type are identical (i.e., they have the same number and type of cores, and the same power consumption characteristics). The characteristic of compute nodes (e.g., number of cores, the power consumption properties, and/or the computational performance of cores) differs across compute node types. The total number of cores in the data center is $\underline{\mathrm{NCORES}}$. We use a global index for cores. Let $\underline{\mathrm{CT}_k}$ be the type of the compute node to which core *k* belongs. Because all cores within a compute node are identical, we also refer to CT_k as being the type of core *k*.

The power consumption of a compute node is the sum of its base power consumption and the power consumption of its cores. The base power consumption is used to model non-compute devices (e.g., disks, fans). The power consumed by the non-compute devices is not affected by the utilization of the cores [MuV07]. Let $\underline{B_j}$ be the base power consumption of a compute node of type *j*. We assume that compute nodes are not turned off during the execution of a workload.

Therefore, the base power consumption will always be incurred even if the compute node is not executing any tasks.

Each core of type j in the data center can be put in one of $\underline{\eta_j}$ P-states. P-state 0 is the P-state with the highest clock frequency and highest power consumption. Each consecutive P-state has a lower clock frequency and lower power consumption. We also consider the case where we can turn off the core. We model the case where the core is turned off by adding one additional P-state to the available P-states of a core. The turned-off P-state will be the highest P-state (e.g., for cores with P-states from P0 to P5, the turned-off state will be P6). The power consumption of a core of type j running in P-state k is $\underline{\pi_{j,k}}$. In some cases, the power consumption of a core is also a function of the task type that it executes. For example, I/O intensive tasks usually consume less power than other tasks [MuV07]. In this work, we assume that the power consumption of a core is dependent on its type and P-state alone. However, it is possible to extend our model to capture the effect of a task type (e.g., I/O or compute intensive task types) on core power consumption. A third index would have to be added to π to represent the effect of a task type on the power consumption of a core.

Let $\underline{\mathrm{PS}_k}$ be the assigned P-state of core k. Let $\underline{\mathrm{cores}_j}$ be the set of indices of cores that belong to compute node j. The power consumption of compute node j, $\underline{\mathrm{PCN}_j}$, is given by:

$$\mathrm{PCN}_j = B_{\mathrm{NT}_j} + \textstyle\sum_{k \in \mathrm{cores}_j} \pi_{\mathrm{NT}_j,\mathrm{PS}_k} \,. \tag{4.1}$$

The first term refers to the baseline power for a compute node j of type NT_j and the second term refers to the active operational power that depends on the P-state.

4.3.4. Estimated Computational Speed

We assume that the estimated computational speed (ECS) of a task of type *i* on a core of type *j* running in P-state *k*, ECS(i, j, k), is known. ECS(i, j, k) represents the number of tasks of type *i* that can be completed per time unit on a core of type *j* when running in P-state *k*. The estimated computational speed is equal to the reciprocal of the estimated time to compute (ETC) [AlM10, AlM11a]. The assumption of ETC information is a common practice in resource allocation research (e.g., [BaS01, DhA02, GhY93, KaA98, KhP93, SiY96, XuN01]). The ETC values for a given system can be obtained from user supplied information, experimental data, or task profiling and analytical benchmarking [AlB05, GhY93, KhP93, XuN01]. Obviously, when the core is turned-off, the ECS of a task of any type is 0, i.e., ECS(i, j, $\eta_j - 1$) = 0 for all *i* and *j*.

4.3.5. Computer Room AC Units

We assume that the number of CRAC units in a data center is NCRAC. CRAC units are used to remove the heat generated by the compute nodes. The power consumed by a CRAC unit is equal to the ratio of the amount of heat removed at that CRAC unit to the Coefficient of Performance (CoP) of that CRAC unit [MoC05].

The amount of heat removed by a CRAC unit *i* depends on the inlet air temperature, $\text{TCRAC}_i^{\text{in}}$ (which is directly affected by the heat generated by compute nodes), and the assigned outlet air temperature, $\text{TCRAC}_i^{\text{out}}$ (which is the temperature of the cool air to be generated by the CRAC). Let ρ be the density of air, Cp be the specific heat capacity of air, and FCRAC_i be the air flow rate at CRAC unit *i*. The amount of heat removed per unit time at CRAC unit *i* is equal to [TaM06]:

$$\rho \cdot \text{Cp} \cdot \text{FCRAC}_i \cdot (\text{TCRAC}_i^{\text{in}} - \text{TCRAC}_i^{\text{out}}). \tag{4.2}$$

The CoP of a CRAC unit is a function of its outlet temperature [MoC05]. The power consumed by CRAC unit *i*, PCRAC$_j$, is given by [MoC05]

$$\text{PCRAC}_i = \frac{\rho\cdot\text{Cp}\cdot\text{FCRAC}_i\cdot\left(\text{TCRAC}_i^{\text{in}}\text{-TCRAC}_i^{\text{out}}\right)}{\text{CoP}\left(\text{TCRAC}_i^{\text{out}}\right)}. \tag{4.3}$$

When the inlet air temperature of a CRAC unit is less than or equal to the assigned outlet temperature (there is no heat to be removed) the power consumption is 0.

4.4. Thermal Constraints

Due to the law of energy conservation, the power consumed at a compute node will be dissipated as heat causing an increase in the temperature of the air going through the compute node. To maintain the reliability of the CRACs and compute nodes, CRAC units must remove the heat generated by the compute nodes so that the inlet air temperature of the CRACs and compute nodes are kept at or below a redline temperature. Let TCN_i^{in} and $\text{TCN}_i^{\text{out}}$ be the inlet and outlet air temperatures at compute node *i*, respectively. Let FCN$_i$ be the air flow rate at compute node *i*. The outlet air temperature of compute node *i* is given by [TaM06]

$$\text{TCN}_i^{\text{out}} = \text{TCN}_i^{\text{in}} + \left(\frac{\text{PCN}_j}{\rho\cdot\text{Cp}\cdot\text{FCN}_i}\right). \tag{4.4}$$

Air flow patterns in data centers are complex. The inlet temperatures of CRAC units and compute nodes are affected by the outlet temperatures of other CRAC units and compute nodes [TaM06]. Let

$$\boldsymbol{T}^{\text{out}}=\left[\text{TCRAC}_1^{\text{out}}, \ldots, \text{TCRAC}_{\text{NCRAC}}^{\text{out}}, \text{TCN}_1^{\text{out}}, \ldots, \text{TCN}_{\text{NCN}}^{\text{out}}\right]^{\text{T}}$$

and

$$\boldsymbol{T}^{\text{in}}=\left[\text{TCRAC}_1^{\text{in}}, \ldots, \text{TCRAC}_{\text{NCRAC}}^{\text{in}}, \text{TCN}_1^{\text{in}}, \ldots, \text{TCN}_{\text{NCN}}^{\text{in}}\right]^{\text{T}}.$$

Using the "Abstract Heat Flow Model" proposed in [TaM06], we can compute each element of $\boldsymbol{T}^{\text{in}}$ as a linear combination of the elements of $\boldsymbol{T}^{\text{out}}$, i.e.,

$$\boldsymbol{T}^{\text{in}} = \text{A}\boldsymbol{T}^{\text{out}}. \tag{4.5}$$

The values in matrix A can be estimated using sensor measurements [TaM06]. Let $\boldsymbol{T}^{\text{redline}}$ be the vector of redline temperature constraints on inlet air temperatures, which gives the constraint

$$\boldsymbol{T}^{\text{in}} \leq \boldsymbol{T}^{\text{redline}}. \tag{4.6}$$

The inequality is element-wise (i.e., element *i* in vector $\boldsymbol{T}^{\text{in}}$must be less than or equal to element *i* in vector $\boldsymbol{T}^{\text{redline}}$)

4.5. Assignment Problems

4.5.1. Overview

Given a workload composed of a set of tasks arriving at different times, the goal of assignment Problem 1 is to maximize the total reward rate subject to a constraint on the maximum total power consumption. The goal of assignment Problem 2 is to minimize the power consumption of the data center subject to a constraint on the minimum reward rate. Both problems are subject to thermal constraints (i.e., the redline temperatures at the inlets must not be exceeded). The decisions that both assignment problems must make are: 1) the P-states of cores, 2) the task to core assignments, and 3) the outlet temperatures of the CRAC units.

Because we assume the arrival rates of task types remain constant, the decision variables will remain constant. If the arrival rates change, then we will have another optimization problem that we will have to solve to obtain the new values of the decision variables.

Temperature evolution in the data center is in orders of minutes, while the execution of a task is in orders of seconds or milliseconds. To make workload assignments tractable, previous research (e.g., [AdV10, PaT10]) has used a two-stage assignment approach. The first stage manages the power and the thermal evolution in the data center, while the second stage performs workload balancing. In this chapter, we apply the two-stage assignment approach for both of our assignment problems. In the first stage, our approach assigns the P-states of cores, the desired execution rate of task types on cores, and the outlet temperatures of CRAC units. The first stage guarantees that the thermal constraints and the power constraint for Problem 1 or the reward constraint for Problem 2 are not violated. In the second stage, our approach implements a dynamic scheduler that assigns tasks to cores so that the actual execution rate of each task type on each core approaches the desired execution rate set by the first step. The dynamic scheduler can also make the decision to drop a task. The two-stage assignment is depicted in Figure 4.2.

In Subsection 4.5.2, we focus on the first-stage assignment problem for solving Problem 1. The difference between the first-stage assignment for Problem 1 and Problem 2 is shown in Subsection 4.5.3. In Subsection 4.5.4, we propose a dynamic scheduler to assign incoming tasks to cores.

4.5.2. First-Stage Assignment: Problem 1

Overview

In the following subsection (problem formulation), we formulate the assignment problem as an exact mixed integer nonlinear program (MINLP). Because the solution techniques for solving the exact MINLP are not scalable, we propose a scalable technique to find near-optimal solutions. The technique divides Stage 1 into three steps. The first step assigns power budgets to

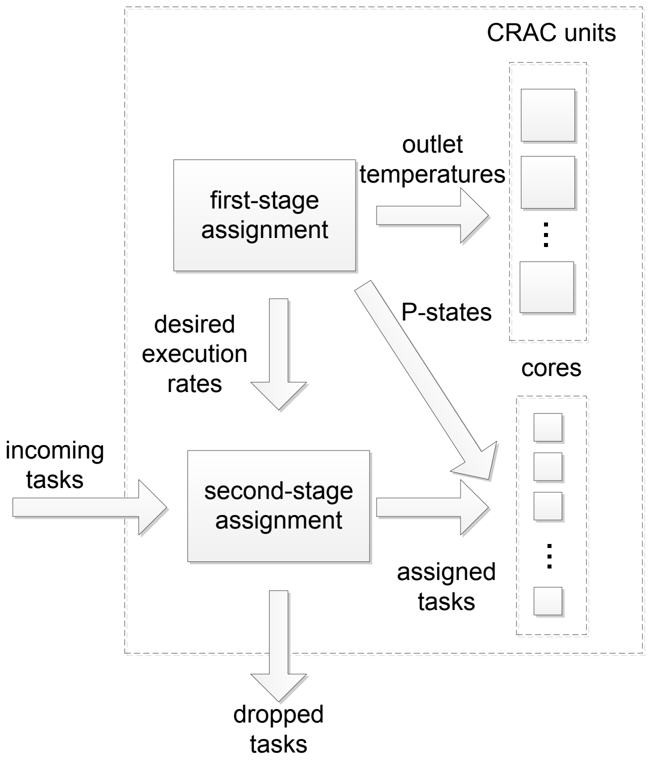


Figure 4.2. The assignment problem in the data center. The first stage assigns the outlet temperatures of CRAC units, the P-states of cores, and the desired execution rate of task types on cores. The second stage assigns the incoming tasks to cores based on the desired execution rate set by the first step or drops tasks that cannot make their deadline.

compute nodes, CRAC outlet temperatures, and the fraction of time each core spends running tasks of each type. The second step converts the power budget assignment into a P-state assignment for each core. Finally, Step 3 uses the exact P-state assignment in Step 2 to assigns the desired execution rate of each task type on each compute node.

Problem Formulation

The decisions made at the first stage are: the outlet temperature of each CRAC unit ($\text{TCRAC}_i^{\text{out}}$**)**, the P-state of each core (PS_k), and the desired rate of executing tasks of each type on each core. The desired rates are organized in a matrix ER. Entry ER(i, k) represents the desired execution rate of tasks of type i on core k. Once a P-state of a core is assigned, we assume that it is not changed.

The following equation shows the assignment problem for Problem 1:

$$\text{maximize } \textstyle\sum_{i=1}^{T}\left(r_i \sum_{k=1}^{\text{NCORES}} \text{ER}(i, k)\right) \tag{4.7}$$

subject to:

1. $\sum_{i=1}^{T} \text{ER}(i, k)/\text{ECS}(i, \text{CT}_k, \text{PS}_k) \le 1$, $k = 1,\ldots, \text{NCORES}$.
2. $\text{ER}(i, k)\ (d_i - (1/\text{ECS}(i, \text{CT}_k,\text{PS}_k))) \ge 0$, $i = 1,\ldots, T$ and $k = 1,\ldots, \text{NCORES}$.
3. $\sum_{k=1}^{\text{NCORES}} \text{ER}(i, k) \le \lambda_i, \quad i = 1,\ldots, T$.
4. $\sum_{j=1}^{\text{NCN}} \text{PCN}_j + \sum_{i=1}^{\text{NCRAC}} \text{PCRAC}_i \le P_{\text{const}}$.
5. $\boldsymbol{T}^{\text{in}} \le \boldsymbol{T}^{\text{redline}}$.

The objective function is the total reward rate. The first constraint guarantees that the desired execution rate of task types on a core will not exceed the core's ability to complete the tasks. When the estimated execution time of a task of type i on a core of type j running in P-state k (i.e., $1/\text{ECS}(i, j, k)$) is greater than d_i, no task of type i can make its deadline on core k even if its execution starts immediately after its arrival. Therefore, if $1/\text{ECS}(i, j, k) > d_i$, then Constraint 2 guarantees that $\text{ER}(i, k) = 0$ to avoid executing tasks of type i on core k. Constraint 3 guarantees that the sum of the desired execution rate of a task type on all cores does not exceed its arrival rate. The power constraint is guaranteed by Constraint 4. Finally, Constraint 5 guarantees the thermal constraints.

Note that there are two cases where ECS values can be 0. First, when PS_k is the turned-off P-state, the ECS of any task type on core k is 0. Second, a core type may not be able to execute certain task types (for example, due to certain required software not being installed on the corresponding node type). When an ECS value is 0, 1/ECS will not be defined. However, we can solve this issue by assuming that the ECS value is a "small enough" positive number.

The problem in Equation 4.7 is a MINLP for the following two reasons. First, the above problem contains integer constraints due to the requirement that the P-states be integers. Second, the measured CoP of the CRAC units at the HP Labs Utility Data Center as a function of CRAC output temperature, τ, is given by [MoC05]

$$\mathrm{CoP}(\tau) = 0.0068\tau^2 + 0.0008\tau + 0.458 \tag{4.8}$$

For this CoP, the power consumption of the CRAC units (Equation 4.3) is nonlinear (and non-convex), which makes constraint 4 a nonlinear (and non-convex) constraint.

MINLPs belong to the class of NP-hard problems. Finding the optimal solution of such problems is computationally infeasible for large problem sizes. For example, consider a compute node that has 32 cores and that each core can be put in one of 5 P-states. This gives us $5^{32} \approx 2.3 \times 10^{22}$ P-state assignment combinations.

In the following subsections, we show how the Stage 1 assignment is divided into three steps to relax the integer P-state constraints. In the first step, instead of assigning P-states to cores, we assign power consumption to cores. This makes the assignment problem simpler. The decision variables in the first step are the power consumption of each compute node, the outlet temperature of each CRAC unit, and the fraction of time each compute node spends on tasks of each type. The second step converts the compute node power assignment into a P-state assignment. The third step assigns the desired execution rate of each task type on each core for the P-state assignment obtained from the second step.

Step 1 Assignment

Relaxing the integer P-state constraint means that we allow a core to be assigned a continuous P-state value rather than a discrete one. Therefore, we have to define core power consumption and ECS functions for continuous P-states. Another equivalent assignment problem is to assume that cores can be assigned a continuous power value between zero and the power consumption in P-state 0. We use this equivalent assignment problem because it makes the representation of the assignment problem easier (we relate power directly with performance), and it eliminates the need to define power consumption functions for continuous P-states.

For the relaxed problem, the ECS of a task of type *i* on core *k* is a continuous function of the power consumption of the core. Let $\underline{\text{PCORE}_k}$ be the power assigned to core *k*. Let $\underline{C}^{\text{ECS}}_{i,j}(\text{PCORE}_k)$ be the ECS for a task of type *i* running on a core of type *j* as a continuous function of PCORE_k. To minimize the difference between the integer solution and the relaxed solution of the Step 1 assignment, we select $C^{\text{ECS}}_{i,j}$ so that it goes through the points

$$(\pi_{j,0}, \text{ECS}(i, j, 0)), \ldots, (\pi_{j,\eta_j-1}, \text{ECS}(i, j, \eta_j - 1)).$$

Each of these points is the power consumption of a P-state and the ECS at that P-state. Intuitively, one can view the value of $C^{\text{ECS}}_{i,j}$ when PCORE_k is not equal to the power consumption of any P-state, is to assume that the core can switch between a P-state with power consumption lower than PCORE_k and a P-state with a power consumption higher than PCORE_k such that the average power consumption is equal to PCORE_k. Therefore, we chose to represent $C^{\text{ECS}}_{i,j}$ using a piecewise linear function.

For example, assume a core of type *j* with four P-states. The power consumption of P-states 0, 1, 2, and 3 is 0.15, 0.1, 0.05, and 0 Watts (W), respectively. The ECS values for task type *i* for each of the four P-states are 1.2, 0.9, 0.5, and 0, respectively. The $C^{\text{ECS}}_{i,j}$ function is a linear piecewise function that goes through the points (0, 0), (0.05, 0.5), (0.1, 0.9), and (0.15, 1.2). This function is shown in Figure 4.3.

Because for both Problems 1 and 2 a higher ECS value will result in a better solution, if the $C^{\text{ECS}}_{i,j}$ function is concave, then the computational expense of the optimization can be greatly reduced by representing the $C_{i,j}^{\text{ECS}}$ function with linear constraints. The $C_{i,j}^{\text{ECS}}$ function, however, is not guaranteed to be concave. In that case, an equivalent representation is only achieved by introducing additional binary constraints. The introduction of the binary constraints would make the Stage 1 optimization problem computationally infeasible for a large number of cores. For instance, consider the example shown in Figure 4.4 where the number of P-states is four (i.e., $\eta_j = 4$). The ECS values for the P-states 0, 1, 2, and 3 are 1.2, 0.9, 0.2, and 0, respectively. This $C_{i,j}^{\text{ECS}}$ function is not a concave function.

The non-concavity of a $C_{i,j}^{\text{ECS}}$ function is caused by a P-state that has an ECS to power consumption ratio that is less than its next lower P-state. We call this P-state a "bad" P-state. For the $C_{i,j}^{\text{ECS}}$ function in Figure 4.4. P-state 2 is a "bad" P-state because the ratio of its ECS to its power consumption is 4, where the ratio of P-state 1's ECS to its power consumption is 9. If we ignore P-state 2 (the "bad" P-state), then the $C_{i,j}^{\text{ECS}}$ function will be concave. This case is shown in Figure 4.5. We ignore "bad" P-states (i.e., do not assign a core a "bad" P-state) so that we can reduce the computational complexity of the Step 1 assignment.

In general, when the "bad" P-states are not ignored, the Step 1 assignment will still avoid "bad" P-states. For example, consider the case where a compute node of type j has two cores. Assume that the compute node can assign a maximum of 0.1 W total power to its cores. Assume that there is only one task type and it has a reward of 1, i.e., $r_i = 1$. If the function in Figure 4.4 is the ECS of that compute node, then the optimal solution in this case would be to put one of the cores in P-state 1 (i.e., assign 0.1 W power to it) and the other in P-state 3 (i.e., assign 0 W power to it). This will result in a total reward rate of 0.45, which is the same as when "bad" P-states are ignored. It should be noted that the optimal value when the "bad" P-states are ignored is never better than the optimal value when the "bad" P-states are *not* ignored.

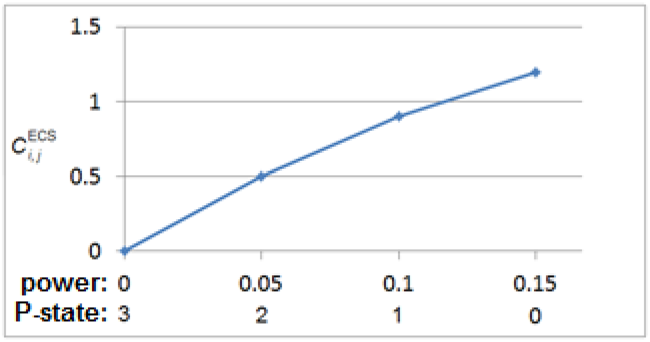


Figure 4.3. An example $C_{i,j}^{ECS}$ function.

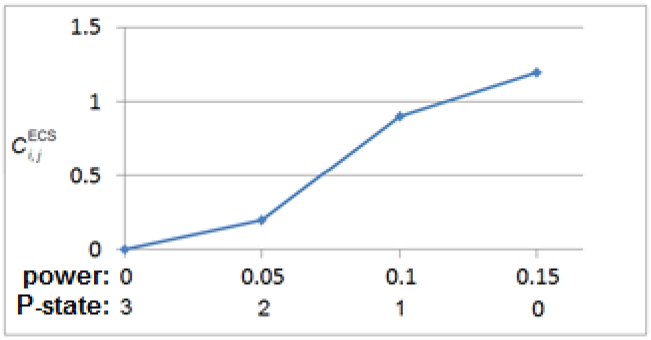


Figure 4.4. An example of a non-concave $C_{i,j}^{ECS}$ function.

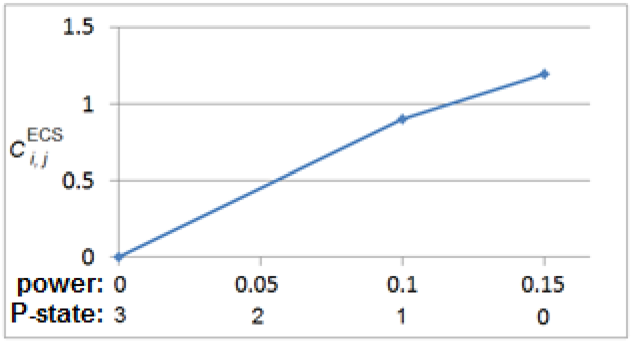


Figure 4.5. An illustration of the calculation of the $C_{i,j}^{ECS}$ function in Figure 4.4 when the "bad" P-state is ignored.

Let DF(i, k) be the desired fraction of time that core k spends executing tasks of type i.

The desired execution rate of task type i on core k, ER(i, k), is equal to

$$\mathrm{DF}(i,k)\, C^{\mathrm{ECS}}_{i,CT_k}\ (\mathrm{PCORE}_k).$$

The Step 1 (relaxed) optimization problem is obtained from Equation 4.7 by replacing $\mathrm{ER}(i, k)$ with

$$\mathrm{DF}(i,k)\, C^{\mathrm{ECS}}_{i,CT_k}\ (\mathrm{PCORE}_k)$$

and $\mathrm{ECS}\big(i, \mathrm{CT}_k, \mathrm{PS}_k\big)$ with $C^{\mathrm{ECS}}_{i,CT_k}$ (PCORE_k). The effect of Constraint 2 is captured by considering a P-state k of core type j a "bad" P-state for task type i if $d_i < 1/\mathrm{ECS}(i, j, k)$. Equation 4.9 and its constraints represent the Step 1 optimization problem.

$$\text{maximize} \sum_{i=1}^{T} \Big(r_i \sum_{k=1}^{\mathrm{NCORES}} \mathrm{DF}\big(i,k\big) C^{\mathrm{ECS}}_{i,k}(\mathrm{PCORE}_k) \Big) \tag{4.9}$$

subject to:

1. $\sum_{i=1}^{T} \mathrm{DF}\big(i,k\big) \le 1,\ k = 1, \ldots, \mathrm{NCORES}.$
2. $\sum_{k=1}^{\mathrm{NCORES}} \mathrm{DF}\big(i,k\big) C^{\mathrm{ECS}}_{i,k}(\mathrm{PCORE}_k) \le \lambda_i,\ i = 1, \ldots, T.$
3. $\sum_{j=1}^{\mathrm{NCN}} \mathrm{PCN}_j + \sum_{i=1}^{\mathrm{NCRAC}} \mathrm{PCRAC}_i \le P_{\mathrm{const}}.$
4. $\boldsymbol{T}^{\mathrm{in}} \le \boldsymbol{T}^{\mathrm{redline}}.$

Because all the cores within a compute node are homogenous, we can reduce the time to find a solution for Equation 4.9 by assuming that all cores within a compute node will be assigned the same value of DF for each task type and the same power consumption (PCORE_k). This reduces the size of matrix DF and the number of ECS functions to be equal to $T \cdot \mathrm{NCN}$.

The problem in Equation 4.9 is an NLP. To avoid locally optimal solutions that have $\sum_{i=1}^{T} \mathrm{DF}\big(i,k\big) < 1$ (i.e., core k is not fully utilized), we substitute the inequality in Constraint 1 with an equality. Even with this change, a solution to Equation 4.9 may be locally (and not globally) optimal. Different locally optimal solutions may be obtained from different initial starting points. Therefore, we try multiple random starting points. Details on how we decided on the number of random start points are in Section 4.7 (simulation results).

Step 2 Assignment

The purpose of the Step 2 assignment is to convert the power assigned to a core *k* into a P-state. The solution to Step 1 guarantees that the power and thermal constraints are satisfied. Therefore, the power consumption at any compute node should be kept at or below the power consumption that resulted from the Step 1 assignment. We design the following heuristic to convert the PCORE_k values into a P-state assignment:

1. For each core *k,* assign it the highest possible P-state that results in a power consumption greater than or equal to PCORE_k.
2. For each compute node *j*

    While the power consumption calculated by Equation 1 is greater than the power consumption that resulted from Step 1

        Increase the P-state of the core with the smallest P-state the next non-bad P-states.

Because the $C_{i,j}^{\text{ECS}}$ functions are concave, the ECS to power consumption ratio of a P-state will always be lower than or equal to that of a higher P-state. Therefore, in Step 2 of the procedure above we increase the P-state of the core with the lowest P-state. In cases where there are multiple task types assigned to a core, we only ignore the bad P-states of the task type that gives the most reward rate on that core.

Step 3 Assignment

In Step 3, we solve Equation 4.7 to determine the optimal desired execution rate of task types on cores (i.e., the optimal ER matrix) using the outlet temperature of CRACs that is determined in Step 1 and the discrete P-state assignment determined in Step 2, which make the solution to Equation 4.7 a simple linear program (LP).

4.5.3. First-Stage Assignment: Problem 2

The exact formulation of the first-stage assignment for Problem 2 is similar to the formulation of Problem 1 (given in Equation 4.7) except that the objective function for Problem 2 is to minimize power consumption subject to a constraint on the minimum total reward rate. The exact formulation of Problem 2 is also a MINLP. Therefore, we propose a three-step approach for solving Problem 2. Similar to the first step of Problem 1, the first step of Problem 2 uses the $C_{i,j}^{\mathrm{ECS}}$ functions to relax the integer P-state assignment constraints.

Step 2 of the first-stage assignment for Problem 2 converts the power consumption assignment of cores into a discrete P-state assignment. To guarantee that the total reward rate constraint is not violated, the conversion at a specific compute node j must guarantee that the cores in their assigned P-states will collectively be capable of executing task types at a rate that is greater than or equal to the total desired execution rate that is set by Step 1 for compute node j.

A simple way to guarantee that the total reward rate constraint is not violated at a compute node j is to assign each core in compute node j to the highest possible P-state that results in power consumption greater than or equal to PCORE_k. This guarantees the total reward rate constraint because we assume that $\mathrm{ECS}(i, j, p) \geq \mathrm{ECS}(i, j, p + 1)$.

One may be able to improve this simple P-state assignment (i.e., reduce the power consumption) by incrementing the P-state of some cores. However, incrementing the P-state of any core will require reassigning the desired execution rate of task types among the compute node cores so that the total reward rate constraint is satisfied. We show how a mixed integer program can be used to find the optimal P-state assignment that satisfies the total reward rate constraints.

Let the reassigned desired execution rates of task types on cores be arranged in matrix $\underline{\text{RER}}$. Entry RER(i, k) represents the reassigned desired execution rate of task type i on core k. For cores in compute node j, the following mixed integer program finds the optimal P-state assignment and reassigned desired execution rates for cores that belong to compute node j:

$$\text{minimize} \sum_{k \in \text{cores}_j} \pi_{\text{NT}_j,\text{PS}_k}, \; k \in \text{cores}_j \tag{4.10}$$

subject to:

1. $\sum_{i=1}^{T} \text{RER}(i,k)/\text{ECS}(i, \text{CT}_k, \text{PS}_k) \le 1, k \in \text{cores}_j$.
2. $\text{RER}(i,k)\,(m_i - (1/\text{ECS}(i, \text{CT}_k, \text{PS}_k)) \ge 0$.
   $i = 1,\dots, T$ and $k \in \text{cores}_j$.
3. $\sum_{k \in \text{cores}_j} \text{RER}(i,k) \ge \sum_{k \in \text{cores}_j} \text{ER}(i,k)$, $i = 1,\dots, T$.

Constraints 1 and 2 are similar to Constraints 1 and 2 in Equation 7. Constraint 3 guarantees that the total available execution rate for each task type at compute node j is greater than or equal to the total desired execution rate set by Step 1. The problem in Equation 4.10 contains integer constraints due to the P-state assignment that makes it computationally intractable for many of today's compute nodes that contain a large number of cores. However, for a fixed P-state assignment, the problem in Equation 4.10 is a LP feasibility problem. We use the LP feasibility problem to test whether a specific P-state assignment satisfies the total reward rate constraints.

We designed the following procedure to convert the power consumption of cores into a P-state assignment:

1. For each core k, assign it the highest possible P-state that results in power consumption greater than or equal to PCORE_k.

2. For each compute node *j*
    a. Until a feasible solution exists for Problem 4.10

        Increase the P-state of the core with the smallest P-state to the next non-bad P-state.

The output of Step 2 is a P-state assignment and the matrix RER (which now becomes the new ER). Similar to Step 2 of Problem 1, Step 2 of Problem 2 avoids "bad" P-states.

The P-state assignment of Step 2 may violate the thermal constraints. Therefore, Step 3 solves the exact optimization problem for Problem 2. Because the P-state and desired execution rates are determined, the exact optimization problem becomes an NLP (due to the power consumption of CRAC units).

4.5.4. Second Stage Assignment

The second stage assignment for both Problems 1 and 2 is the same. The dynamic scheduler at the second step keeps track of the actual execution rate of each task type on each core in matrix $\underline{\text{AER}}$. The goal of the dynamic scheduler is to make the ratio of AER(*i, k*)/TC(*i*, *k*) as close as possible to 1 for each task type *i* and core *k*.

For each incoming task *t,* the dynamic scheduler maps *t* to a core that can complete it before its deadline and has the minimum AER(*i*,*k*)/TC(*i*,*k*) value that is less than or equal to 1. If no such core exists, then the dynamic scheduler drops *t*.

4.6. Simulation Setup

4.6.1. Overview

We conducted simulation studies to evaluate the effectiveness of our assignment technique. In this section, we show how the parameters of the simulations were generated.

4.6.2. Compute Nodes

In our simulations, we used a varying number of compute nodes. Each compute node belongs to one of two compute node types based on two 7U servers listed in the SPECpower_ssj2008 results [Spe08]. The first compute node type is based on the HP ProLiant DL785 G5 server. This server has eight AMD Opteron 8381 HE processors with four cores in each processor. The second compute node type is based on the NEC Express5800/A1080a-S server. This server has four Intel Xeon X7560 processors. Each processor has eight cores. Table 4.1 lists the parameters of both node types. Details on how the values of the parameters were obtained are in Appendix C.

We used a uniform random variable to assign a node type to compute nodes. Each compute node type has an equal probability of being assigned to a compute node.

4.6.3. ECS Matrices

The estimated computation speed values are arranged in a three-dimensional ECS matrix. These dimensions represent task types, node types, and P-states. In a real world environment, the ECS values can be based on user supplied information, experimental data, or task profiling and analytical benchmarking (e.g., [AlB05, FrS93, GhY93, KhP93, MaB99, YaA93, YaK94]). The following paragraphs discuss how we generated synthetic ECS data for the purposes of our simulations.

In our simulations, we have eight task types, two compute node types, and four P-states (not including the turned-off P-state). First, we generate a two dimensional ECS matrix for P-state 0. The columns represent the node types and the rows represent the task types. The ratio of the performance of node type 1 to node type 2 is 0.6 (this is based on the number of server side Java operations per second each node type can perform [Spe08]). Therefore, we assumed that the

TABLE 4.1. PARAMETERS OF THE TWO NODE TYPES USED IN SIMULATIONS

node type	**1**	**2**
base power consumption (kW)	0.353	0.418
number of cores	32	32
number of P-states	4	4
power consumption of P-state 0 (kW)	0.01375	0.01625
clock frequencies of P-states (MHz)	2500, 2100, 1700, 800	2666, 2200, 1700, 1000
air flow rate (m^3/s)	0.07	0.0828

average ECS over all task types for node types 1 and 2 is 0.6 and 1, respectively. Let NTYPES be the number of compute node types in the data center. The easiness of a task type i , TE_i, is assumed to be equal to the sum of the ECS values over all node types for P-state 0, i.e., $\text{TE}_i = \sum_{j=1}^{\text{NTYPE}} \text{ECS}(i, j, 0)$. For illustration purposes, we assume that the easiness for task type i is half that of task type i + 1. Let rand[a, b] be a uniform random variable in the interval [a, b]. Entry (i, j) in the two-dimensional ECS for P-state 0 is the product of the average ECS for task type i, the average ECS for node type j, and a variation factor rand[$1 - V_{\text{ECS}}$, $1+V_{\text{ECS}}$]. The variation factor is used so that the affinity between task types and node types differs across the types. The value of V_{ECS} that we used is 0.1.

Let $f_{j,k}$ be the clock frequency of a core of type j running in P-state k. In many cases, the ECS of task types on cores is not exactly proportional to clock frequency. For example, reducing the clock frequency will have less impact on the ECS of a task that is I/O bound versus a task that is CPU bound. Therefore, we use a parameter V_{prop} so that the ECS of a task type on a core type is not exactly proportional to the clock frequency of the P-states. The ECS is extended in the third dimension by using

$$\mathrm{ECS}(i, j, k) = \mathrm{ECS}(i, j, 0) \cdot \frac{f_{j,k}}{f_{j,0}} \cdot \mathrm{rand}\left[1 - V_{\mathrm{prop}},\, 1 + V_{\mathrm{prop}}\right]. \tag{4.11}$$

We used two different values for V_{prop} in different sets of simulations, i.e., 0.1 and 0.3.

Using Equation 4.11 may result in a P-state that has a higher ECS value for a specific task type and a specific core type than a lower P-state. To prevent this case, if an entry (i, j, k) is higher than entry $(i, j, k - 1)$, then we generate a random number $\mathrm{rand}\left[1 - V_{\mathrm{prop}}, 1 + V_{\mathrm{prop}}\right]$ for $\mathrm{ECS}(i, j, k)$ until it is less than $\mathrm{ECS}(i, j, k - 1)$. We start by generating the ECS for P-state 1 then P-state 2 and so on.

4.6.4. Task Types

The number of task types in all of our simulations is assumed to be eight. The easiness of a task type i, TE_i, is assumed to be equal to its average ECS value over node types. The reward for completing a task of type i by its deadline is assumed to be equal to the reciprocal of its easiness, i.e.,

$$r_i = 1/\mathrm{TE}_i. \tag{4.12}$$

The above reward value is used for the purposes of our simulations. Note that the reward value is not required to be equal to the reciprocal of task easiness for our assignment technique to work.

Now we show how the d_i values that are used to calculate the deadline of individual tasks are generated. Let $\underline{\mathrm{MinECS}_i}$ and $\underline{\mathrm{MaxECS}_i}$ be the minimum and maximum ECS values for task type i over all core types and all P-states except the turned-off P-state. MinECS_i is given by

$$\mathrm{MinECS}_i = \min\left[\mathrm{ECS}\left(i, j, \eta_j - 2\right)\right] \; 1 \le j \le \mathrm{NTYPES}. \tag{4.13}$$

MaxECS_i is given by

$$\mathrm{MaxECS}_i = \max[\mathrm{ECS}(i, j, 0)] \; 1 \le j \le \mathrm{NTYPES}. \tag{4.14}$$

The value of d_i is given by

$$d_i = 1.5 \text{ rand}[1/\text{MaxECS}_i,\ 1/\text{MinECS}_i]. \tag{4.15}$$

We used Equation 4.15 to compute d_i because it guarantees that there is at least one core type that can make the deadline of a task of type *i*. There is also a chance of generating a task type such that some of its tasks' deadlines can be met by all core types running at their lowest frequency.

The last parameter that needs to be generated for a task type is its arrival rate, λ_i. Let $\underline{\text{SumECS}_i}$ be the ECS obtained for a task type *i* if all cores in the data center are used equally by every task type and all cores are running in P-state 0. The value of SumECS_i is given by

$$\text{SumECS}_i = \textstyle\sum_{k=1}^{\text{NCORES}} \text{ECS}(i, \text{CT}_k, 0)/T. \tag{4.16}$$

Our goal is to assign arrival rates for task types such that the data center can complete all the arriving tasks when running at full capacity (i.e., all cores in P-state 0) but would be oversubscribed if there is a power constraint (i.e., there is not enough power to run all cores in P-state 0). This is not simple to achieve. Therefore, we use SumECS_i to approximate the arrival rates. In addition, to introduce some randomness in the assigned arrival rate of task type *i*, we use a parameter, $\underline{V_{\text{arrival}}}$. Once the arrival rate for a task type is assigned, it remains constant. The arrival rate of task type *i* is given by

$$\lambda_i = \text{SumECS}_i \cdot \text{rand}[1 - V_{\text{arrival}}, 1 + V_{\text{arrival}}]. \tag{4.17}$$

The value of V_{arrival} that we used is 0.3.

4.6.5. Cross Interference Coefficients

In [TaM06], for two compute nodes *i* and *j*, the <u>cross interference coefficient</u>, $\underline{\alpha_{i,j}}$, is the percentage of air recirculated from compute node *i* to compute node *j*. Computational Fluid

Dynamics (CFD) simulations were used in [TaM06] to obtain cross interference coefficients for a small data center (ten racks with five compute nodes in each rack, and one CRAC unit). The time consumed for a single run of a CFD simulation was about an hour with a CFD simulation required for each of the 50 compute nodes [TaM06]. In our simulations, we use 150 compute nodes and three CRAC units. The amount of time to run the CFD simulations for each data center in our simulations is prohibitive. In Appendix D, we show how an LP feasibility problem can be used to generate the cross interference coefficients. Our goal is not to propose a method of calculating the cross interference coefficients for a real data center. Rather, our goal is to generate cross interference coefficients for simulation studies that are based on realistic information about the air flows in data centers.

4.6.6. Power and Thermal Constraints

To set a reasonable power constraint in our simulations for Problem 1, we need to find the minimum and maximum power consumption of the data center. The minimum power consumption occurs when all cores in the data center are turned off. The maximum power consumption occurs when all cores are running in P-state 0. The minimum and maximum power consumption of the data center can be found using the NLP problem below solved for the two extreme values of PCN_j. The solution for this problem will provide the power consumption bounds of the data center. The decision variables are the outlet temperatures of CRAC units. Because it is an NLP problem, our solution to the problem will not necessarily provide the global minimum. Therefore, the solutions are considered an upper bound of the minimum and maximum power consumption of the data center.

$$\text{minimize}\left[\sum_{j=1}^{\text{NCN}} \text{PCN}_j + \sum_{i=1}^{\text{NCRAC}} \text{PCRAC}_i\right] \qquad (4.18)$$

subject to

$$\boldsymbol{T}^{\text{in}} \leq \boldsymbol{T}^{\text{redline}}$$

Let $\underline{P}_{\text{min}}$ and $\underline{P}_{\text{max}}$ be the upper bounds on the minimum and maximum power consumption of the data center, respectively. Let $\underline{\Phi}$ be a "power multiplier" that takes values in the interval [0, 1]. The power multiplier allows us to select a power constraint that is between the minimum and the maximum power consumption bounds. The power constraint, P_{const}, is given by

$$P_{\text{const}} = P_{\text{min}} + (P_{\text{max}} - P_{\text{min}}) \cdot \Phi \tag{4.19}$$

The redline inlet air temperature was set at 25° Celsius for compute nodes and 40° Celsius for CRAC units.

4.6.7. Total Reward Constraint

To set a reasonable total reward rate constraint for Problem 2, we need to find the maximum possible total reward rate that occurs when all cores are running in P-state 0. Let $\underline{\text{FRAC}}(i, j)$ be the average fraction of a core in compute node j that is used to execute task of type i. The effective number of cores at compute node j that are used to execute tasks of type i, $\underline{E}(i, j)$, is equal $|\text{cores}_j|\text{FRAC}(i, j)$. The maximum reward rate can be found using the following LP:

$$\text{maximize} \sum_{i=1}^{T} \sum_{j=1}^{\text{NCN}} r_i \text{ECS}(i, j, 0) E(i, j) \tag{4.20}$$

subject to

1. $\sum_{j=1}^{\text{NCN}} \text{ECS}(i, j, 0) E(i, j) \leq \lambda_i,\ 1 \leq i \leq T.$
2. $\sum_{i=1}^{T} E(i, j) \leq |\text{cores}_j|,\ \ 1 \leq j \leq \text{NCN}.$
3. $\boldsymbol{T}^{\text{in}} \leq \boldsymbol{T}^{\text{redline}}.$

4.6.8. CRAC units

In our simulations, we assumed that there are three homogeneous CRAC units. The CoP for each CRAC unit is given by Equation 4.8. The air flow rate of each CRAC unit is set so that the sum of the air flow rates of the compute nodes is equal to the sum of the flow rates of the CRAC units. The layout of the data center is given in Figure 4.1.

4.7. Simulation Results

4.7.1. Comparison Overview

One may choose to run all cores in the data center in P-state 0 without considering the power consumption implications. Although this approach is simple and will result in the highest reward rate, it may violate the power constraint and result in a lower reward rate per power consumption. We show this in the next subsection.

We performed simulations for the first-stage assignment problem and compared our technique with a technique that only considers putting a core in P-state 0 or turning off the core. The authors in [PaT10] show how the fraction of the computational resources at a compute node can be used to compute the power consumption of a compute node and the QoS obtained from that compute node. Our techniques solve different assignment problems than the technique in [PaT10] and our techniques consider P-state assignments. We adapt the technique in [PaT10] by relating the fraction of compute resources (cores) used at a compute node to the reward rate obtained from that compute node and the total power consumed at that node as described by Equations 4.21 and 4.22. We compare our techniques with the one described in Equation 22.

The power consumption of compute node j is given by

$$\mathrm{PCN}_j = B_i + \pi_{\mathrm{NT}_j,0} \sum_{i=1}^{T} E(i, j) \tag{4.21}$$

The reward rate for a task of type i at compute node j is equal to $r_i\text{ECS}(i, j, 0)E(i, j)$. The comparison technique for solving Problem 1 is given by

$$\text{maximize} \sum_{i=1}^{T} \sum_{j=1}^{\text{NCN}} r_i \text{ECS}(i, j, 0) E(i, j) \tag{4.22}$$

subject to

1. $\sum_{i=1}^{T} E(i, j) \leq |\text{cores}_j|, \ 1 \leq j \leq \text{NCN}.$
2. $\sum_{j=1}^{\text{NCN}} \text{ECS}(i, j, 0) E(i, j) \leq \lambda_i, 1 \leq i \leq T.$
3. $\sum_{j=1}^{\text{NCN}} \text{PCN}_j + \sum_{i=1}^{\text{NCRAC}} \text{PCRAC}_i \leq P_{\text{const}}.$
4. $\boldsymbol{T}^{\text{in}} \leq \boldsymbol{T}^{\text{redline}}.$.

The effective number of cores used at a compute node must not exceed the total number of cores at that compute node. This is guaranteed by Constraint 1. Constraint 2 guarantees that the execution rate for a task type is not higher than its arrival rate. Constraints 3 and 4 are the power and thermal constraints, respectively. The deadline constraints can be dealt with by setting $E(i, j) = 0$ whenever $d_i < (1/\text{ECS}(i, \text{NT}_j, 0))$. Equation 4.22 is an NLP problem due to the power consumption of CRAC units. The comparison technique for solving Problem 2 is similar to the comparison technique for solving Problem 1 except that the objective is to minimize power consumption while guaranteeing that the total reward rate does not drop below R_{const}.

The problem formulation in Equation 4.22 (based on [PaT10]) is similar to our Step 1 formulation in Equation 4.9. However, Equation 4.22 only considers P-state 0 whereas Equation 4.9 considers multiple P-states including P-state 0. Therefore, Equation 4.9 considers a superset of the P-state assignment options compared to Equation 4.22. Therefore, the optimal assignment of Equation 4.9 should be better than or equal the optimal assignment of Equation 4.22. Two examples where the assignments of Equations 4.9 and Equation 4.22 have the same objective value are: a) when cores in the data center have only one P-state (i.e., P-state 0), and b) when the all P-states except P-state 0 are "bad" P-states.

4.7.2. Results

Overview

We have conducted simulations to compare our technique against the one described in Equation 4.22. We illustrate the effect of the following three parameters on the performance: 1) static power consumption of cores, 2) the variation of the ECS values from being proportional to the clock frequency of cores, and 3) the reward and power constraints. Note that the static power consumption is part of the total power consumption of a *core* and is different than the base power consumption of a *compute node*. The static power consumption is part of the second term in Equation 1. The total power consumption of a compute node is equal to the sum of its base power consumption and the sum of the static and dynamic power consumption of its cores. The results in this section show the percentage increase in the reward rate or the percentage decrease in the power consumption that our technique achieves in comparison to the one described in Equation 4.22.

Random Starting Points

Because the Step 1 assignments of both Problems 1 and 2 are NLPs, their solutions may be locally optimal. The quality of the locally optimal solution is affected by the starting point of the NLP optimization. To determine an appropriate number of starting points to use, we have conducted 20 simulations, each using 100 randomly generated starting points. For each simulation, we determined the number of random starting points needed to obtain a solution that is within 1% of the best solution. The upper limit of a 95% confidence interval of the number of solutions was 11.45, so we used 11 random starting points for our simulations and one starting point that is the solution of Equation 4.22.

The Problem in Equation 4.22 is a NLP due to the power consumption of the CRAC units. Therefore, the solutions to the problems may be locally optimal. A brute force discretized optimization of a problem that has three CRAC units, 150 compute nodes, and eight task types, is computationally intractable. However, tests on smaller problems, i.e., two CRAC units, 40 compute nodes, and eight task types, have shown no improvement. Therefore, we only use a single starting point to find the solution to Equation 4.22.

Main Results for Problems 1 and 2

Figures 4.6 and 4.7 show the percentage increase in the reward rate for Problem 1 and the percentage reduction in power consumption for Problem 2 that our technique achieves. Each bar in Figures 4.6 and 4.7 represents the average of 20 simulations. Error bars are added to show a 95% confidence interval around the average.

Both figures show that as the static power consumption of P-state 0 increases, the relative performance of our technique decreases. Because P-state 0 runs at a higher voltage and frequency, the percentage of dynamic power consumption is usually higher than that of the other P-states. Therefore, the static power as a percentage of the overall power consumption for the other P-states will be higher compared to that of P-state 0. The static power consumption is not related to the frequency, so the higher P-states will have a lower performance (in terms of clock frequency) to power consumption ratio compared to P-state 0. When the performance to power consumption ratio of P-state 0 is the highest among all the P-states, the assignment technique of Equation 4.22 will perform as well as our technique. The static power percentages for all the P-states of each node type are shown in Table 4.2. The static power consumption of the P-states in each node type is calculated using the static power of P-state 0. For our simulations, we assumed three different static power consumption percentages for P-state 0 (10, 20, and 30%). Refer to appendix C for details about the calculation of the static power.

Figures 4.6 and 4.7 also show that the relative performance of our technique increases as V_{prop} increases from 0 to 0.3. For a given core type, a higher value of V_{prop} gives a higher affinity of P-states to task types (i.e., some P-states will be better suited for specific task types than others). Therefore, more reward rate per power consumption can be obtained by matching task types with their better suited P-states.

The reason that our technique achieves higher increase in reward rate for Problem 1 compared to the decrease in power consumption for Problem 2 is that there is a lower bound on

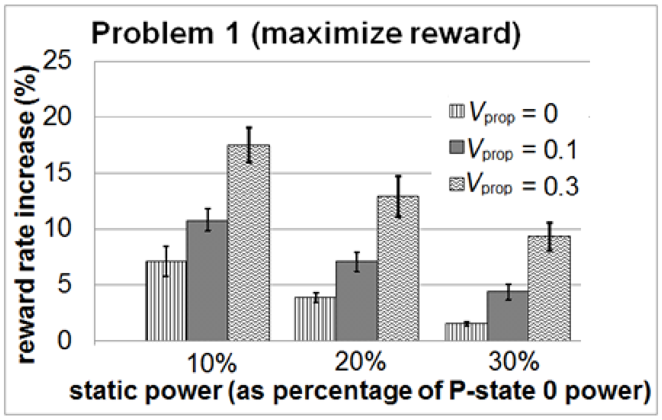


Figure 4.6. This figure shows the results for Problem 1 (maximizing reward rate). The average percentage improvement obtained by using the three-step assignment given in Section 4.5.2 versus the assignment that is based on [PaT10] (given in Equation 4.22) is shown. A 95% confidence interval is shown for each average percentage improvement. The static power consumption of P-state 0 as a percentage of the total processing core power consumption is increased from 10% to 30%. Each group of columns compares the results when the value of V_{prop} is 0, 0.1, and 0.3. The number of compute nodes, task types, and CRAC units for each simulation is 150, eight, and three, respectively.

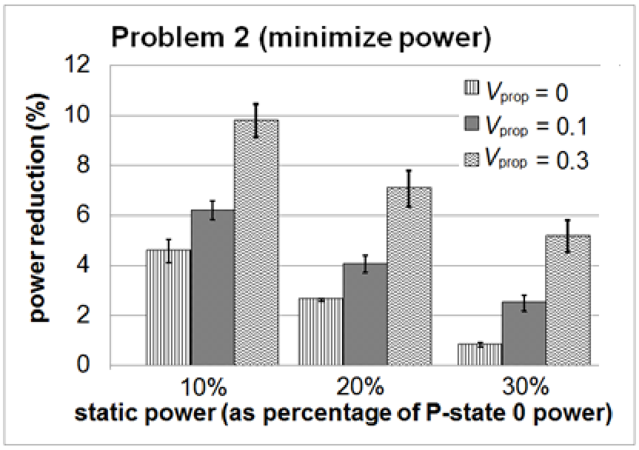


Figure 4.7. This figure shows the results for Problem 2 (minimizing power consumption). The average percentage improvement obtained by using the three-step assignment given in Section 4.5.3 versus the assignment that is based on [PaT10] (given in Equation 4.22) is shown. A 95% confidence interval is shown for each average percentage improvement. The static power consumption of P-state 0 as a percentage of the total processing power consumption is increased from 10% to 30%. Each group of columns compares the results when the value of V_{prop} is 0,0.1, and 0.3. The number of compute nodes, task types, and CRAC units for each simulation is 150, eight, and three, respectively.

TABLE 4.2. STATIC POWER CONSUMPTOIN OF P-STATES OF CORES IN EACH NODE TYPE AS A PERCENTAGE.

	node type 1 (P-state 1, 2, 3)	node type 2 (P-state 1, 2, 3)
P-state 0 static power =10%	12.3, 15.6, 31.0%	12.5, 16.6, 27.5%
P-state 0 static power =20%	24.0, 29.3, 50.2%	24.4, 31.0, 46.0%
P-state 0 static power =30%	35.1, 41.5, 63.4%	35.6, 43.5, 59.4%

the minimum power. The lower bound occurs when all cores are turned off and all compute nodes are only consuming the base power. However, the minimum reward rate of the data center is zero, which also happens when all cores are turned off. Because the minimum power is greater than zero, Problem 2 leaves less opportunity for improvement than Problem 1. If we do not consider the minimum power consumption of the data center for both our technique and the technique in Equation 4.22, then percentage power reduction that our technique achieves over the technique in Equation 22 will be on average 1.58 times higher on average.

The simple approach of running all cores in the data center at P-state 0 will result in a violation of the power constraint for Problem 1 and higher power consumption for Problem 2. For example, when P-state 0 static power is 10% of its total power consumption and V_{prop} is 0.1, the simple approach resulted in a violation of the power constraint by 42% for Problem 1 and a power consumption of 75% higher than our approach for Problem 2.

Effect of Power and Reward Constraints

We also have conducted simulations to show the effect of increasing the power and reward constraints on the performance of our techniques. For these simulations, the static power consumption of P-state 0 was set to 10% of its total power consumption and V_{prop} was set to 0.3. These simulations are shown in Figures 4.8 and 4.9.

As the power constraint gets tighter (i.e., the power multiplier value gets lower) the relative performance of our technique improves. This is because when power is scarce, managing it intelligently can lead to substantial performance gains. As the power constraint gets looser, our technique will start assigning lower P-states to take advantage of the available power. Therefore, the performance of our technique will be closer to the performance of the technique in Equation 4.22 until they are both equal when the power multiplier is equal to 1.

As shown in Figure 4.9, when the reward constraint is low, the relative performance of our technique is low. This is because there is a lower bound on the minimum power consumption of the data center. However, as the reward constraint increases, the power needed to satisfy the reward rate constraint becomes higher and more power savings can be obtained using our technique. Even when the reward rate constraint is at 90% of the maximum possible, our technique achieves 8.6% improvement. When the reward rate constraint is 100% of the maximum possible, both our technique and the one in Equation 4.22 will run all cores in the data

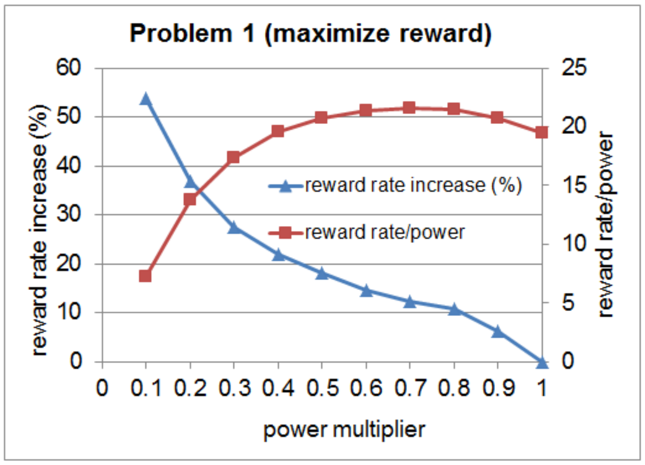


Figure 4.8. This figure shows the percentage increase in reward rate that our approach archives over the technique in Equation 4.22 and the reward rate per power consumption for our technique for Problem 1 (maximizing reward rate). The power multiplier is increased from 0.1 to 1 with a step of 0.1. The static power of P-state 0 is 10% and V_{prop} is 0.3. Each point in the figure represents a simulation case for one data center. The number of compute nodes, task types, and CRAC units for each simulation is 150, eight, and three, respectively.

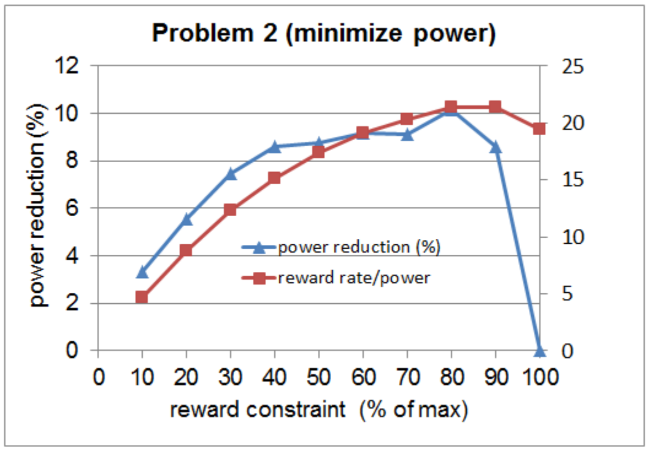


Figure 4.9. This figure shows the percentage reduction in power consumption that our approach archives over the technique in Equation 4.22 and the reward rate per power consumption for our technique for Problem 2 (minimizing power). The reward rate as a percentage of the maximum possible reward rate is increases from 10% to 100%. The static power of P-state 0 is 10% and V_{prop} is 0.3. Each point in the figure represents a simulation case for one data center. The number of compute nodes, task types, and CRAC units for each simulation is 150, eight, and three, respectively.

center at P-state 0 to satisfy the constraint. Therefore, our approach will have no reward rate improvements. Figures 4.8 and 4.9 also show the reward rate per power consumption ratio for different power and reward rate constraints. When the power multiplier is low (i.e., tighter power constraint) or the reward rate constraint is low (i.e., looser reward rate constraint) both solutions to Problem 1 and Problem 2 are driven to consume less power and collect less reward. Because of the minimum power consumption of the data center is not zero, the ratio of the reward rate to power consumption *decreases* more with the reduction in reward rate than it does *increase* with the reduction in total power consumption. This explains the low reward rate to power consumption ratio when the power multiplier is low or the reward rate constraint is low.

There are two reasons that cause the ratio of reward rate to power consumption to decline for a higher value of a power constraint for Problem 1 or a higher value of a reward rate constraint for Problem 2. The first reason is that our technique will run the cores at lower P-states that are not power efficient so that all the power that is available is used (in Problem 1) or the reward rate constraint is satisfied (in Problem 2). The second reason is that our technique will assign tasks of types that have low reward rates because all the higher reward rate tasks are fully assigned. The simple approach of running all cores in the data center at P-state 0 will be equivalent to the case where the reward rate constraint is 100% of the maximum and the case where the power multiplier is 1. The simple approach will result in a reward rate per power consumption ratio equal to 19.43 that is less than the highest ratio which is equal to 21.6 for Problem 1(Figure 4.8) and 21.4 for Problem 2 (Figure 4.9).

Scalability Analysis

In our approach, the step that consumes the most time is Step 1. We conducted a scalability analysis for the execution time of Step 1 for Problem 1 (the execution time for Problem 2 was similar to Problem 1). Table 4.3 shows the execution times in seconds for different numbers of compute nodes. The number of task types remains fixed at eight task types. In Table 4.4, we have increased the number of task types as the number of compute nodes was increased. The execution times for each case in both Tables 4.3 and 4.4 are averaged across five simulation runs. All the simulations were run on a laptop computer with a Core i7 processor running at a clock frequency of 2.8 GHz. The number of CRAC units was fixed at three CRAC units.

Tables 4.3 and 4.4 show that the execution time is sensitive to both the number of task types and the number of compute nodes. Recall that the task type arrival rate is a function of the number of cores and the number of task types (see Equation 16). As the number of task types increases the arrival rate of each task type decreases. However, the total workload remains constant.

Because the calculation of Step 1 is done offline (i.e., the assignment decisions are made before tasks arrive) based on estimated task arrival rates obtained from historical data for each task type, it is feasible to execute it for a longer time compared to online techniques. Furthermore, if Step 1 is parallelized (for example, by calculating the solutions to multiple starting points in parallel), then the time consumed by it can be significantly reduced.

Unexecuted Workload

Because of the power constraint in Problem 1 and the reduction of power consumption in Problem 2, a portion of the workload will not be executed. Tables E.1 and E.2 in Appendix E give the percentages of unexecuted tasks of each type for our technique versus the technique in Equation 4.22.

Two cases for each combination of static power consumption and V_{prop} are given. Table E.1 shows the percentage of unexecuted tasks for Problem 1 and Table E.2 shows the percentage of unexecuted tasks for Problem 2.

The task types that are not executed are the ones with low reward per power consumption

TABLE 4.3. EXCUTION TIMES FOR STEP 1 OF PROBLEM 1 AS THE NUMBER OF COMPUTE NODES VARIES.

number of nodes	number of task types	average execution time (seconds)
30	8	42
90	8	679
150	8	2771
210	8	3688
270	8	9575

TABLE 4.4. EXCUTION TIMES FOR STEP 1 OF PROBLEM 1 AS THE NUMBER OF COMPUTE NODES AND THE NUMBER OF TASK TYPES VARY.

number of nodes	number of task types	average execution time (seconds)
30	2	10
90	5	181
150	8	2771
210	11	12242
270	14	134842

ratio. In many cases, these task types were the same for our technique and the technique in Equation 4.22. However, there are some cases were this was not the case. This is because our technique considers higher P-states that may have a better reward per power consumption for different task type than the technique in Equation 4.22.

Both our technique and the technique in Equation 4.22 execute more tasks for Problem 1 compared to Problem 2. This is because in Problem 1 the goal is to execute as many tasks as possible to maximize the reward. However, in Problem 2 the goal is to minimize the power consumption which will result in less tasks being executed because we are not concerned with collecting reward at a rate higher than the reward rate constraint.

Summary

Our results show an average improvement over the comparison technique of up to 17% for Problem 1 (maximizing reward) and up to 9% for Problem 2 (minimizing power). Higher percentage increase in reward rate can be achieved for data centers with tighter power constraints. Because today's data centers are large, these improvements can mean hundreds of thousands of dollars in additional revenue or power savings. For example, the average cost of electricity in the U.S. for the industrial sector is \$0.0688/kWh [Eia12]. If we achieve 9% power savings in a data center that has an average power consumption of 5 MW, then that will result in about \$271,395 in annual savings.

4.8. Future Work

In this study, we selected a reward value for task types to be the reciprocal of the task easiness. Therefore, easier tasks will have lower reward. In future work, we will study other possible task type rewards and their effect on the improvements that our technique can achieve.

4.9. Conclusions

In this chapter, we study two assignment problems. The first problem maximizes the reward collected for completing tasks by their deadlines with a constraint on the maximum total power consumption. The second problem minimizes power consumption with a constraint on the minimum reward rate. We show how the P-states can be assigned at the data center level and divide each assignment problem into two stages. The first stage assigns the P-states of cores, the desired number of tasks per unit time allocated to a core, and the outlet CRAC temperatures. The second stage assigns individual tasks as they arrive at the data center to cores so that the actual number of tasks per unit time allocated to a core approaches the desired number set by the first stage.

We formulate the first-stage assignment as a MINLP. Because the MINLP is not scalable with respect to the number of cores, we propose a multi-step, scalable assignment technique. At the second stage, we propose a dynamic scheduler to assign tasks entering the data center to cores.

In many data centers where static and dynamic core power consumptions are considered, P-state 0 is not the P-state with the highest performance to power consumption ratio. Therefore, using the assignment techniques in this chapter will result in a better total reward over a technique that choses between putting a core in P-state 0 or turning it off.

We conducted simulations to show the effectiveness of our technique over a technique based on [PaT10] which did not consider multiple P-states. In some cases, our technique achieved 17% average improvement for the problem of maximizing reward and 9% average improvement for the problem of minimizing the power consumption. In a large data center, these improvements can mean hundreds of thousands of dollars in additional revenue from additional productivity (Problem 1) or power savings (Problem 2).

Chapter 5

Conclusions

This thesis describes two problems. The first problem is how to quantify heterogeneity of computing environments. In Chapter 2, we proposed three statistical measures to quantify the heterogeneity. We demonstrated the impact that the heterogeneity measures may have on the performance of task assignment heuristics and how these measures can be used to select a task assignment heuristic that is best suited for a computing environment based on its heterogeneity.

In Chapter 3, we identified three properties that heterogeneity measures should have. Using these properties, we proposed three heterogeneity measures: (a) machine performance homogeneity, (b) task easiness homogeneity, and (c) task-machine affinity. We demonstrated how the SVD is used to calculate the task-machine affinity. We illustrated how an iterative row and column normalization procedure can be used to keep the measures independent (an important property that a heterogeneity measures should have).

The second problem that we study in this research is power and thermal-aware workload allocation in constrained data centers. In Chapter 4, we described two workload allocation (assignment) problems. The first allocation problem is concerned with maximizing the reward collected for completing tasks by their individual deadline subject to a constraint on the maximum total power consumption. The second allocation problem is concerned with minimizing the total power consumption of a data center subject to a constraint on the minimum total reward rate.

In the simulations that we conducted, our technique achieved up to 17% average increase in reward rate for the problem of maximizing reward and 9% average decrease in power consumption for the problem of minimizing the power consumption compared to a technique that considers only running a core in P-state 0 or turning-off the core. In a large data centers, these improvements can mean hundreds of thousands of dollars in additional revenue from improved productivity (Problem 1) or power savings (Problem 2).

References

[AbV10] Z. Abbasi, G. Varsamopoulos, and E. K. S. Gupta, "Thermal aware server provisioning and workload distribution for internet data centers," *19th ACM International Symposium on High Performance Distributed Computing (HPDC'10)*, 2010, pp. 130-141.

[AlB05] S. Ali, T. D. Braun, H. J. Siegel, A. A. Maciejewski, N. Beck, L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, "Characterizing resource allocation heuristics for heterogeneous computing systems," *Advances in Computers*, Vol. 63: Parallel, Distributed, and Pervasive Computing, 2005, pp. 91–128.

[AlK08] S. Ali, J. K. Kim, H. J. Siegel, and A. A. Maciejewski, "Static heuristics for robust resource allocation of continuously executing applications," *Journal of Parallel and Distributed Computing*, Vol. 68, No. 8, Aug. 2008, pp. 1070–1080.

[AlM10] A. M. Al-Qawasmeh, A. A. Maciejewski, and H. J. Siegel, "Characterizing heterogeneous computing environments using singular value decomposition," 19th Heterogeneity in Computing Workshop (HCW 2010), *24th International Parallel and Distributed Processing Symposium, Workshops and PhD Forum (IPDPSW 2010)*, Apr. 2010.

[AlM11a] A. M. Al-Qawasmeh, A. A. Maciejewski, and H. J. Siegel, "Characterizing task-machine affinity in heterogeneous computing envinronments," 20th Heterogeneity in Computing Workshop (HCW 2011), *25th International Parallel and Distributed Processing Symposium, Workshops and PhD Forum (IPDPSW 2011)*, Apr. 2011, pp. 34-44.

[AlM11b] A. M. Al-Qawasmeh, A. A. Maciejewski, H. Wang, J. Smith, H. J. Siegel, and J. Potter, "Statistical measures for quantifying task and machine heterogeneities," *The Journal of Supercomputing*, Special Issue on Advances in Parallel and Distributed Computing, Vol. 57, No. 1, pp. 34-50, July 2011

[AlP12] Abdulla M. Al-Qawasmeh, Sudeep Pasricha, Anthony A. Maciejewski, and Howard Jay Siegel, "Thermal-Aware Performance Optimization in Power Constrained Heterogeneous Data Centers," 21st Heterogeneity in Computing Workshop (HCW 2012), *2012 International Parallel and Distributed Processing Symposium Workshops and PhD Forum (IPDPSW)*, pp. 27 – 40, May 2012.

[AlS00] S. Ali, H. J. Siegel, M. Maheswaran, D. Hensgen, and S. Ali., "Representing task and machine heterogeneities for heterogeneous computing systems," *Tamkang Journal of Science and Engineering*, Special 50th Anniversary Issue, Vol. 3, No.3, Nov. 2000, pp.195–207.

[Amd10] AMD Family 10h Server and Workstation Processor Power and Thermal Data Sheet. Publication # 43374, Revision 3.19, June 2010.

[ApY11] J. Apodaca, D. Young, L. Briceno, J. Smith, S. Pasricha, A. A. Maciejewski, H. J. Siegel, S. Bahirat, B. Khemka, A. Ramirez, and Y. Zou, "Stochastically robust static resource allocation for energy minimization with a makespan constraint in a heterogeneous computing environment," *9th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA '11)*, Dec. 2011, 10 pp.

[ArH98] R. Armstrong, D. Hensgen, and T. Kidd, "The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions," *7th IEEE Heterogeneous Computing Workshop (HCW '98)*, 1998, pp. 79–87.

[AyM01] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Dynamic and aggressive scheduling techniques for power-aware real-time systems," *22nd IEEE Real-Time Systems Symposium (RTSS '01)*, Dec. 2001, pp. 95-105.

[BaS01] H. Barada, S. M. Sait, and N. Baig, "Task matching and scheduling in heterogeneous systems using simulated evolution," 10th Heterogeneous Computing Workshop (HCW 2001), *15th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2001)*, Apr. 2001.

[BaV01] I. Banicescu and V. Velusamy, "Performance of scheduling scientific applications with adaptive weighted factoring," 10th Heterogeneous Computing Workshop (HCW 2001), *15th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2001)*, Apr. 2001.

[BeA12] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing," Future Generation Computer Systems, Vol. 28, No. 5, pp. 755-768, May 2012.

[BrF84] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*, Wadsworth, Belmont, CA, 1984.

[BrR10] D. Brown and C. Reams, "Toward energy-efficient computing," *Communications of the ACM*, Vol. 53, No. 3, Mar. 2010, 14 pp.

[BrS01] T. D. Braun, H. J. Siegel, N. Beck, L. Bölöni, R. F. Freund, D. Hensgen, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, Vol. 61, No. 6, June 2001, pp. 810–837.

[BrS08] T. D. Braun, H. J. Siegel, A. A. Maciejewski and Y. Hong, "Static mapping heuristics for tasks with dependencies, priorities, deadlines, and multiple versions in heterogeneous environments," *Journal of Parallel and Distributed Computing*, Vol. 68, No. 11, Nov. 2008, pp. 1504–1516.

[BrS11] L. Briceno, H. J. Siegel, A. Maciejewski, M. Oltikar, J. Brateman, J. White, J. Martin, and K. Knapp, "Heuristics for Robust Resource Allocation of Satellite Weather Data Processing onto a Heterogeneous Parallel System," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 22, No. 11, Nov. 2011, pp. 1780 - 1787.

[BuP97] A. Burns, S. Punnekkat, B. Littlewood, and D. Wright, *Probabilistic Guarantees for Fault Tolerant Real-Time Systems*, Technical Report Design for Validation (DeVa) TR No. 44, Esprit Long Term Research Project No. 20072, Department of Computer Science, University of Newcastle upon Tyne, UK, 1997.

[BuS00] J. A. Butts and G. S. Sohi, "A static power model for architects," *33rd annual ACM/IEEE international symposium on Microarchitecture (MICRO 33)*, Dec. 2000, pp. 191-201.

[CaJ09] L. Canon and E. Jeannot, Precise Evaluation of the Efficiency and the Robustness of Stochastic DAG Schedules, Research Report 6895, INRIA, April 2009.

[CaR05] K. W. Cameron, R. Ge, and X. Feng, "High-performance, power-aware distributed computing for scientific applications," *IEEE Computer*, Vol. 26, No.11, Nov. 2005, pp. 40- 47.

[ChG10] J. Choi, S Govindan, J. Jeong, B. Urgaonkar, and A. Sivasubramaniam, "Power consumption prediction and power-aware packing in consolidated environments," *IEEE Transactions on Computers*, Vol. 59, No. 12, Dec. 2010.

[ChM10] R. C. Chiang, A. A. Maciejewski, A. L. Rosenberg, and H. J. Siegel, "Statistical predictors of computing power in heterogeneous clusters," 19th Heterogeneity in Computing Workshop (HCW 2010), *24th International Parallel and Distributed Processing Symposium, Workshops and PhD Forum (IPDPSW 2010)*, Apr. 2010.

[Cof76] E. G. Coffman, ed., *Computer and Job-Shop Scheduling Theory*, John Wiley & Sons, New York, NY, 1976.

[CoR10] M. Couceiro, P. Romano, and L. Rodrigues, "A machine learning approach to performance prediction of total order broadcast protocols," *4th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, 2010.

[DhA02] M. K. Dhodhi, I. Ahmad, and A. Yatama, "An integrated technique for task matching and scheduling onto distributed heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, Vol. 62, Sep. 2002, pp. 1338–1361.

[DiC01] Q. Ding and G. Chen, "A benefit function mapping heuristic for a class of meta-tasks in grid environments," *CCGRID '01: 1st International Symposium on Cluster Computing and the Grid*, May 2001.

[Eia12] http://www.eia.gov/electricity/data.cfm. Last accessed 4/26/2012.

[ElK02] E. N. Elnozahy, M. Kistler, and R. Rajamony, "Energy-efficient server clusters," *Second International Workshop on Power-Aware Computer Systems, 2002*, pp. 179-197.

[Epa07] Environmental Protection Agency. *Report to Congress on Server and Data Center Energy Efficiency*, 2007. http://www.energystar.gov/ia/partners/prod_development/downloads /EPA_Datacenter_Report_Congress_Final1.pdf. Last accessed 12/6/2011.

[EsA09] B. Eslamnour, S. Ali, "Measuring robustness of computing systems," *Simulation Modelling Practice and Theory*, Vol. 17, No. 9, Oct. 2009, pp. 1457–1467.

[Fer89] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Transactions on Software Engineering*, Vol. 15, No. 11, Nov. 1989, pp. 1427–1436.

[FiH08] D. Filani, J. He, S. Gao, M. Rajappa, A. Kumar, P. Shah, and R. Nagappan, "Dynamic data center power management: Trends, issues, and solutions," *Intel Technology Journal*, Vol. 12, No. 1, 2008, pp. 59-67.

[FrS93] R. F. Freund and H. J. Siegel, "Heterogeneous processing," *IEEE Computer*, vol. 26, June 1993, pp.13–17.

[GhM05] S. Ghanbari and M. R. Meybodi, "On-line mapping algorithms in highly heterogeneous computational grids: A learning automata approach," *International Conference on Information and Knowledge Technology (IKT '05)*, May 2005.

[GhY93] A. Ghafoor and J. Yang, "A distributed heterogeneous supercomputing management system," *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 78–86.

[GoV89] G. H. Golub and C. F. Van Loan, *Matrix Computations, Second Edition,* The John Hopkins University Press, Baltimore, MD, 1989.

[Gub06] J. A. Gubner, *Probability and Random Processes for Electrical and Computer Engineering*, Cambridge University Press, Cambridge, NY, 2006.

[Hew10] Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., and Toshiba Corporation Std., *Advanced Configuration and Power Interface Specification*, Rev. 4.0a, Apr. 2010, http://www.acpi.info/DOWNLOADS/ACPIspec40a.pdf. Last accessed 1/3/2012.

[HuY09] D. Huang, Y. Yuan, L. Zhang, and K. Zhao, "Research on tasks scheduling algorithms for dynamic and uncertain computing grid based on a+bi connection number of SPA," *Journal of Software*, Vol. 4, No 10, Dec 2009, pp. 1102–1109.

[IbK77] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on nonidentical processors," *Journal of the ACM*, Vol. 24, No. 2, Apr. 1977, pp. 280–289.

[JaM11] J.-W. Jang, M. Jeon, H.-S. Kim, H. Jo, J.-S. Kim, and S. Maeng, "Energy reductions in consolidated servers through memory-aware vitual machine scheduling," *IEEE Transactions on Computers*, Vol. 60, No. 4, Apr. 2011.

[JiL05] Z. Jinquan, N. Lina, and J. Changjun, "A heuristic scheduling strategy for independent tasks on grid," *8th International Conference on High-Performance Computing in Asia-Pacific Region 2005*, Nov. 2005.

[KaA98] M. Kafil and I. Ahmad, "Optimal task assignment in heterogeneous distributed computing systems," *IEEE Concurrency*, Vol. 6, No. 3, July 1998, pp. 42–51.

[KiS08] J.-K. Kim, H. J. Siegel, A. A. Maciejewski, and R. Eigenmann, "Dynamic resouce management in energy constrained heterogeneous computing systems using voltage scalling," *IEEE Transactions on Parallel and Distributed Systems*, Special Issue on Power-Aware Parallel and Distributed Systems, Vol. 19, No. 11, pp. 1445-1457, Nov. 2008.

[KaU07] K. Kaya, B. Ucar, and C. Aykanat, "Heuristics for scheduling file-sharing tasks on heterogeneous systems with distributed repositories," *Journal of Parallel and Distributed Computing*, vol. 67, no. 3, Mar. 2007, pp. 271–285.

[KhA06] S. U. Khan and I. Ahmad, "Non-cooperative, semi-cooperative, and cooperative games-based grid resource allocation," *20th International Parallel and Distributed Processing Symposium (IPDPS 2006)*, Apr. 2006.

[KhP93] A. Khokhar, V. K. Prasanna, M. E. Shaaban, and C. Wang, "Heterogeneous computing: Challenges and opportunities," *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 18–27.

[KiH06] J.-K. Kim, D. A. Hensgen, T. Kidd, H. J. Siegel, D. S. John, C. Irvine, T. Levin, N. W. Porter, V. K. Prasanna, and R. F. Freund, "A flexible multi-dimensional QoS performance measure framework for distributed heterogeneous systems," *Cluster Computing*, Special Issue on Cluster Computing in Science and Engineering, Vol. 6, No. 3, July 2006, pp. 281–296.

[LeP95] C. Leangsuksun, J. Potter, and S. Scott, "Dynamic task mapping algorithms for a distributed heterogeneous computing environment," *4th IEEE Heterogeneous Computing Workshop (HCW 1995)*, Apr. 1995, pp. 30–34.

[LiH06] Y. Lin and L. He, “Dual-Vdd interconnect with chip-level time slack allocation for FPGA power reduction,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 25, No. 10, Oct. 2006, pp. 2023 – 2034.

[LoS01] J. R. Lorch and A. J. Smith, “Improving voltage scaling algorithims with PACE,” *ACM SIGMETRICS Performance Evaluation Review*, Vol. 29, No. 1, June 2001, pp.50 -61.

[MaA99] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, “Dynamic mapping of a class of independent tasks onto heterogeneous computing systems,” *Journal of Parallel and Distributed Computing*, Vol. 59, No. 2, Nov. 1999, pp. 107–121.

[MaB99] M. Maheswaran, T. D. Braun, and H. J. Siegel, “Heterogeneous distributed computing,” *Encyclopedia of Electrical and Electronics Engineering*, vol. 8, JohnWiley, NY, 1999, pp. 679–690.

[MaO68] A. W. Marshall and I. Olkin, “Scaling of Matrices to Achieve specified row and column sums,” *Numerische Mathematik*, Vol. 12, No. 1, 1968, pp. 83-90.

[MeS07] A. M. Mehta, J. Smith, H. J. Siegel, A. A. Maciejewski, A. Jayaseelan, and B. Ye, “Dynamic resource allocation heuristics that manage tradeoff between makespan and robustness,” *Journal of Supercomputing*, Special Issue on Grid Technology, Vol. 42, No. 1, pp. 33–58, 2007.

[MiF00] Z. Michalewicz and D. B. Fogel, eds., *How to Solve It: Modern Heuristics*, Springer-Verlag, New York, NY, 2000.

[MoC05] J. Moore, J. Chase, P. Ranganathan, and R. Sharma, "Making scheduling "cool": Temperature-aware workload placement in data centers," *USENIX Annual Technical Conference (ATEC '05)*, 2005, 14 pp.

[MuV07] T. Mukherjee, G. Varsamopoulos, S. K. S. Gupta, and S. Rungta, "Measurement-based power profiling of data center equipment," *IEEE Internatinoal Conference on Cluster Computing*, Sep. 2007, pp. 476 – 477.

[PaG10] E. Pakbaznia, M. Ghasemazar, and M. Pedram, "Temperature-aware dynamic resource provisioning in a power-optimized datacenter," *The Conference on Design, Automation and Test in Europe 2010*, 2010, pp. 124-129.

[PaS06] V. Pallipadi and A. Starikovsky, "The on-demand governor," *The 2006 Linux Symposium*, 2006, pp. 215-229.

[PaT10] L. Parolini, N. Tolia, B. Sinopoli, and B. H. Krogh, "A cyber-physical systems approach to energy management in data centers," *1st ACM/IEEE International Conference on Cyber-Physical Systems*, 2010, pp. 168-177.

[PoC08] J. Poe, C.-B. Cho, and T. Li, "Using analytical models to efficiently explore hardware transactional memory and multi-core co-design," *20th International Symposium on Computer Architecture and High Performance Computing*, 2008.

[PrF88] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C*, Cambridge University Press, Cambridge, NY, 1988.

[ShC09] V. Shestak, E. Chong, A. A. Maciejewski, and H. J. Siegel, "Robust sequential resource allocation in heterogeneous distributed systems with random compute node failures," 18th Heterogeneity in Computing Workshop (HCW 2009), *23rd IEEE*

International Parallel and Distributed Processing Symposium (IPDPS 2009), May 2009.

[Sin64] R. Sinkhorn, “A relationship between arbitrary positive matrices and doubly stochastic matrices,” *The Annals of Mathematical Statistics*, Vol. 35, No. 2 Jun. 1964, pp. 876-879.

[SiY96] H. Singh and A. Youssef, “Mapping and scheduling heterogeneous task graphs using genetic algorithms,” *5th IEEE Heterogeneous Computing Workshop (HCW ’96)*, 1996, pp. 86–97.

[Spe06] Standard performance evaluation corporation (SPEC), Benchmarks, http://www.spec.org, accessed January 12, 2011.

[TaM06] Q. Tang, T. Mukherjee, S. K. S. Gupta, and P. Cayton, “Sensor-based fast thermal evaluation model for energy efficient high-performance datacenters,” *4th International Conference on Intelligent Sensing and Information Processing (ICISIP 2006)*, Dec. 2006, pp. 203-208.

[ToW09] N. Tolia, Z. Wang, P. Ranganathan, C. Bash, and M. Marwah, “Unified thermal and power management in server enclosures,” *InterPACK ’09*, July 2009, 10 pp.

[WuS00] M. Wu, W. Shu, and H. Zhang, “Segmented min-min: A static mapping algorithm for meta-tasks on heterogeneous computing systems,” *9th IEEE Heterogeneous Computing Workshop*, Mar. 2000, pp. 375–385.

[WuS01] M. Wu and W. Shu, “A high-performance mapping algorithm for heterogeneous computing systems,” *15th International Parallel and Distributed Processing Symposium (IPDPS 2001)*, Apr. 2001.

[XiL08] C. Xian, Y.-H. Lu, and Z. Li, “Dynamic voltage scaling for multitasking real-time systems with uncertain execution time,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 27, No. 8, Aug. 2008, pp. 1467-1478.

[XuN01] D. Xu, K. Nahrstedt, and D. Wichadakul, “QoS and contention-aware multi-resource reservation,” *Cluster Computing*, Vol. 4, No. 2, Apr. 2001, pp. 95–107.

[YaA93] J. Yang, I. Ahmad, and A. Ghafoor, “Estimation of execution times on heterogeneous supercomputer architectures,” *International Conference on Parallel Processing,* vol. I, Aug. 1993, pp. 219–225.

[YaK94] J. Yang, A. Khokhar, S. Sheikh, and A. Ghafoor, “Estimating execution time for parallel tasks in heterogeneous processing (HP) environment,” *Heterogeneous Computing Workshop*, pp., 23-28, Apr. 1994.

[YoA11] B. Dalton Young, Jonathan Apodaca, Luis Diego Briceño, Jay Smith, Sudeep Pasricha, Anthony A. Maciejewski, Howard Jay Siegel, Bhavesh Khemka, Shirish Bahirat, Adrian Ramirez, and Young Zou “Deadline and Energy Constrained Dynamic Resource Allocation in a Heterogeneous Computing Environment,” *The Journal of Supercomputing*, accepted, to appear.

[YuV08] H. Yu, B. Veeravalli, and Y. Ha, “Dynamic scheduling of imprecise-computation tasks in maximizing QoS under energy constraints for embedded systems,” *2008 Asia and South Pacific Design Automation Conference (ASPDAC ‘08)*, Mar. 2008, pp. 452-455.

Appendix A

Theorem 1 can be restated using standard matrix notation as:

Theorem A. *Let A be an $m \times n$ matrix with positive elements and let k be a given nonzero scalar. Then there exists an $m \times m$ diagonal matrix D_1 and an $n \times n$ diagonal matrix D_2 such that the matrix D_1AD_2 has the property that each of its rows sums to nk and each of its columns sums to mk. Furthermore, D_1 and D_2 are unique up to a scalar multiple.*

Proof. For simplicity, we will prove the result for a 2×3 matrix A. The general $m \times n$ case readily follows using the same approach. Consider a 3×2 array of A matrices given by the partitioned matrix

$$\widetilde{A} = \begin{bmatrix} A & A \\ A & A \\ A & A \end{bmatrix}.$$

Applying Sinkhorn's theorem to this 6×6 positive matrix, one obtains 6×6 diagonal matrices E and F such that

$$\widetilde{M} \triangleq E\widetilde{A}F = \begin{bmatrix} \widetilde{M}_{11} & \widetilde{M}_{12} \\ \widetilde{M}_{21} & \widetilde{M}_{22} \\ \widetilde{M}_{31} & \widetilde{M}_{32} \end{bmatrix}$$

is a positive matrix whose rows and columns sum to 1. The diagonal matrices E and F can be written as $E = \text{diag}(E_1, E_2, E_3)$ and $F = \text{diag}(F_1, F_2)$ where E_1, E_2, and E_3 are 2×2 diagonal matrices and F_1 and F_2 are 3×3 diagonal matrices. Consider now the 6×3 matrix

$$\overline{M} = \begin{bmatrix} \overline{M}_1 \\ \overline{M}_2 \\ \overline{M}_3 \end{bmatrix} = \begin{bmatrix} \widetilde{M}_{11} + \widetilde{M}_{12} \\ \widetilde{M}_{21} + \widetilde{M}_{22} \\ \widetilde{M}_{31} + \widetilde{M}_{32} \end{bmatrix} = \begin{bmatrix} \widetilde{M}_{11} \\ \widetilde{M}_{21} \\ \widetilde{M}_{31} \end{bmatrix} + \begin{bmatrix} \widetilde{M}_{12} \\ \widetilde{M}_{22} \\ \widetilde{M}_{32} \end{bmatrix}$$

where $\overline{M}_i = \widetilde{M}_{i1} + \widetilde{M}_{i2}$ for $i = 1, 2, 3$. Like $\widetilde{M}$, the six rows of $\overline{M}$ each sum to 1 but the three columns of $\overline{M}$ each sum to 2 instead as each column of $\overline{M}$ is a sum of two columns of $\widetilde{M}$. Next, consider the 2×3 matrix $M = \overline{M}_1 + \overline{M}_2 + \overline{M}_3$. The column sums of M match the column sums of $\overline{M}$ while the row sums of M are given by adding the corresponding rows sums of $\overline{M}_1$, $\overline{M}_2$, and $\overline{M}_3$. Hence, each row of M has a sum of 3 and each column has a sum of 2. Next, note that $M = \sum_{i=1}^{3} \sum_{j=1}^{2} \widetilde{M}_{ij}$. Because $\widetilde{M}_{ij} = E_i A F_j$, we have

$$M = \sum_{i=1}^{3} \sum_{j=1}^{2} E_i A F_j = \left(\sum_{i=1}^{3} E_i \right) A \left(\sum_{j=1}^{2} F_j \right).$$

Setting $D_1 = \sum_{i=1}^{3} E_i$ and $D_2 = \sum_{j=1}^{2} F_j$ results in two diagonal matrices D_1 and D_2 such that $M = D_1 A D_2$ has row sums equal to 3 and column sums equal to 2. Lastly, scaling either D_1 or D_2 by k gives the desired result. The fact that $E = \mathrm{diag}(E_1, E_2, E_3)$ and $F = \mathrm{diag}(F_1, F_2)$ are unique up to a scalar multiple implies the same for D_1 and D_2. Extending the same approach to the $m \times n$ case is now obvious. ■

Appendix B

As in Appendix A, Theorem 2 is restated using standard matrix notation.

Theorem B. *Let A be an m × n matrix with non-negative elements with the property that each row sums to r and each column sums to c. Then mr = nc, and the largest singular value of A is* $\sqrt{rc}$ *with a corresponding input singular vector* $\boldsymbol{v} = 1/\sqrt{n}\,[1 \cdots 1]^T$ *and output singular vector* $\boldsymbol{u} = 1/\sqrt{m}\,[1 \cdots 1]^T$*, respectively. Furthermore, if the matrix is scaled so that* $r = \sqrt{n/m}$ *and* $c = \sqrt{m/n}$*, then the largest singular value is equal to 1.*

Proof. The equality *mr* = *nc* follows from the fact that both values are equal to the sum of the *mn* elements of *A*. The fact that $\sqrt{rc}$ is a singular value with corresponding input and output singular vectors $\boldsymbol{v} = 1/\sqrt{n}\,[1 \cdots 1]^{\mathrm{T}}$ and $\boldsymbol{u} = 1/\sqrt{m}\,[1 \cdots 1]^{\mathrm{T}}$, respectively, readily follows from the facts that *mr* = *nc*, $A\boldsymbol{v} = \sqrt{rc}\boldsymbol{u}$, and $A^{\mathrm{T}}\boldsymbol{u} = \sqrt{rc}\boldsymbol{v}$.

To show that $\sqrt{rc}$ is in fact the largest singular value, suppose that $\boldsymbol{w}$ is an input singular vector associated with the largest singular value. This means that $\boldsymbol{w}$ is a unit vector that maximizes $||A\boldsymbol{w}||$ over the set of all unit *n*-vectors.

We claim that we can assume that the components of $\boldsymbol{w}$ are non-negative. Obviously, we can assume that at least one component of the unit vector *w* is positive since either $\boldsymbol{w}$ or $-\boldsymbol{w}$ can serve as the input singular vector. Suppose that $\boldsymbol{w}$ has at least one positive and at least one negative component. Since *A* is a matrix with non-negative elements, each component of the vector $A\boldsymbol{w}$ is a non-negative combination of the components of $\boldsymbol{w}$. Hence, changing the signs of all the negative components of $\boldsymbol{w}$ will not decrease the magnitudes of the individual components of $A\boldsymbol{w}$ and will not change the norm of $\boldsymbol{w}$. We thus obtain another unit vector $\boldsymbol{z}$, whose components are equal to the absolute values of the corresponding components of $\boldsymbol{w}$, with the property that $||A\boldsymbol{z}|| \geq ||A\boldsymbol{w}||$. By the assumption that $||A\boldsymbol{w}||$ is the largest value over all unit n-vectors, we conclude that the value $||A\boldsymbol{z}|| = ||A\boldsymbol{w}||$ is the largest singular value of A and that the resulting vector $\boldsymbol{z}$ is an input singular vector associated with the largest singular value. We then take the resulting vector $\boldsymbol{z}$ as our new vector $\boldsymbol{w}$ so that the components of $\boldsymbol{w}$ are non-negative.

Now if the largest singular value is different than $\sqrt{rc}$, then the input singular vectors $\boldsymbol{w}$ and $\boldsymbol{v}$ must be orthogonal. However, this is impossible as the dot product of $\boldsymbol{w}$ and $\boldsymbol{v}$ is clearly positive. We thus conclude that $\sqrt{rc}$ is the largest singular value. The last part of the theorem is obtained by applying an appropriate scaling to the matrix A. ■

Appendix C

In this appendix, we show how the parameters for two compute node types used in the simulations (Table 4.1) are obtained. The first compute node type is based on the HP ProLiant DL785 G5 server. This server has eight AMD Opteron 8381 HE processors with four cores in each processor. We assumed that the power consumption of the processor is 0.055 kW. This is based on the thermal design power (TDP) numbers found in the AMD data sheet [*Amd10*]. At 100% utilization, the power consumption of the server was 0.793 kW. To obtain the base power consumption (B_1) of this server, we subtracted the total power consumed by the 8 processors from the power consumption at 100%. The base power consumption is equal to 0.353 kW.

The AMD Opteron processor has 4 P-states. The frequencies of P-states 0, 1, 2, and 3 are 2500, 2100, 1700, and 800 MHz, respectively. The supply voltages of the P-states are 1.325, 1.25, 1.175, and 1.025 V, respectively. To obtain P-state 0 power consumption of an individual core, the total power consumption of the processor is divided by the number of cores. Therefore, the power consumption of P-state 0 is 0.01375 kW. The power consumption of a core is due to static and dynamic power consumption. In [BuS00], the authors show that the static power consumption for a core is equal to a constant multiplied by the supply voltage. Let β be this constant. The dynamic power is calculated by the standard CMOS dynamic power dissipation formula. If S is the number of transistor switches per clock cycle, C_L is the capacitive load, $f_{j,k}$ is the clock frequency of a core of type j running in P-state k and $V_{j,k}$ is the supply voltage of a core of type j running in P-state k, then the dynamic power consumption of a core of type j running in P-state k is equal to $S \cdot C_L \cdot f_{j,k} \cdot V_{j,k}^2$. We assume that $S \cdot C_L$ is a constant and is not dependent on the P-state. Let <u>SC</u> = $S \cdot C_L$. The total core power consumption, $\pi_{j,k}$, is given by

$$\pi_{j,k} = \mathrm{SC} \cdot f_{j,k} \cdot V_{j,k}^2 + \beta \cdot V_{j,k}. \qquad \text{(C.1)}$$

In our simulations, we assume that the static power consumption as a percentage of the total power consumption for P-state 0 is known. Therefore, SC and β in Equation C.1 can be calculated for each core type and the power consumption of the other P-states can be calculated by substituting SC, β, the frequency of each P-state, and the supply voltage of each P-state.

The air flow rate at node type 1 is assumed to be $0.07\text{m}^3/s$. This will guarantee that the maximum increase in temperature of the air going through a node of type 1 will be 9.4° Celsius or 17° Fahrenheit. We assume that the air density is equal to 1.205 kg/m^3 and the specific heat capacity of air is 1 (in reality, the density of air and its specific heat capacity depend on multiple factors such as pressure and temperature).

The second node type is an NEC Express5800/A1080a-S server. This server has four Intel Xeon X7560 processors. Each processor has eight cores. The frequency of P-state 0 is 2666 MHz. We assumed there are four P-states and the frequencies of P-states 1, 2, and 3 are 2200, 1700, and 1000 MHz. The supply voltage for P-states 0 is assumed to be 1.35 V (this is based on Intel Xeon E7540 processor that has the same feature size). The voltages of P-states 1, 2, and 3 are assumed to be 1.268, 1.18, 1.056 V, respectively. The calculation of the other parameters of the second node type is similar to calculation of the parameters of the first node type.

Appendix D

In this appendix, we describe how an LP feasibility problem can be used to generate the cross interference coefficients. As opposed to [TaM06], in our simulations the data centers contain more than one CRAC unit. Therefore, the cross interference coefficients must include the CRAC units in addition to the compute nodes. We assume that the first NCRAC i and j indices in $\alpha_{i,j}$ are CRAC units. If $i \leq$ NCRAC, then $\alpha_{i,j}$ is the percentage of air flow generated from CRAC unit i, otherwise, $\alpha_{i,j}$ is the percentage of the air flow generated from compute node i – NCRAC. Similarly, if $i \leq$ NCRAC, then $\alpha_{i,\,j}$ is the air flow recirculated into CRAC unit j, otherwise, $\alpha_{i,\,j}$ is the air flow recirculated into compute node j – NCRAC.

The exit coefficient $\underline{EC_i}$ is the percentage of air flow of compute node i that goes into CRAC units [TaM06]. Recirculation coefficient $\underline{RC_i}$ is the percentage of the air flow at the inlet of compute node i that is recirculated from the outlet of other compute nodes [TaM06]. The position of a compute node within a rack affects its EC and RC. Compute nodes at the bottom of a rack will have a low RC (most if its inlet air comes from the perforated tiles) and a low EC (most of its air is recirculated into compute nodes). Compute nodes at the top of the rack have a high EC and a high RC. Compute nodes in [TaM06] are labeled A-E. Node A is at the bottom of the rack and node E is at the top of the rack. Table D.1 shows the ranges of EC and RC for each node label based on the CFD simulations in [TaM06].

Because we have more than one CRAC unit in our simulations, for each compute node, we will have an exit coefficient per CRAC unit. Let $\underline{EC_{i,j}}$ be the exit coefficient of compute node i for CRAC unit j.

Usually a data center is arranged in a hot aisle/cold aisle fashion. Figure 4.1 shows the data center layout that we assumed for our simulations. Assuming that CRAC unit i faces hot-aisle i, the EC for a compute node whose outlet air goes into hot-aisle i will have a greater EC to CRAC unit i than to any other CRAC unit. Let $\underline{\text{MinEC}}_l$ and $\underline{\text{MaxEC}}_l$ be the minimum and maximum EC for a compute node with label l (as shown in Table D.1). Let $\underline{\text{MinRC}}_l$ and $\underline{\text{MaxRC}}_l$ be the minimum and maximum RC for a compute node with label l (as shown in Table D.1). Let $\underline{L}_j$ be the label of compute node j. Let $\underline{\text{HA}}_j$ be the hot-aisle to which compute node j belongs (i.e., compute node j's outlet air goes into hot-aisle HA_j). For simplicity, we assume that $\underline{\text{M}}(i, j)$ is the percentage of the EC of a compute node in hot aisle i that goes to CRAC unit j. Let F be the vector of air flow rates F = [FCRAC_1, …, $\text{FCRAC}_{\text{NCRAC}}$, FCN_1, …, FCN_{NCN}]$^{\text{T}}$. Let $\boldsymbol{\alpha}$ be the matrix of the cross interference coefficients.

The LP feasibility problem for generating the cross interference coefficients is given by

Find $\boldsymbol{\alpha}$

subject to

1. $\sum_{i=1}^{\text{NCRAC+NCN}} \alpha_{i,j} = 1,\ 1 \le j \le \text{NCRAC+NCN}$

2. $\sum_{j=1}^{\text{NCRAC+NCN}} \alpha_{i,j} F_i = F_j,\ 1 \le\ i \le \text{NCRAC+NCN}$

3. $\text{MinEC}_{L_i}\text{M}\left(\text{HA}_i, j\right) \le \alpha_{i+\text{NCRAC},j},\ 1 \le i \le \text{NCN}$ and $1 \le j \le \text{NCRAC}$

4. $\alpha_{i+\text{NCRAC},j} \le \text{MaxEC}_{L_i}\text{M}\left(\text{HA}_i, j\right),\ 1 \le i \le \text{NCN}$ and $1 \le j \le \text{NCRAC}$

5. $\text{MinRC}_{L_j} \le \sum_{i=1}^{\text{NCN}} \alpha_{i+\text{NCRAC},j+\text{NCRAC}} \le \text{MinRC}_{L_j},\ 1 \le j \le \text{NCN}$

TABLE D.1. THE RANGES OF EC AND RC FOR DIFFERENT COMPUTE NODE LABELS.

Label	**EC range**	**RC range**
A	30-40%	0-10%
B	30-40%	0-20%
C	40-50%	10-30%
D	70-80%	30-70%
E	80-90%	40-80%

6.

The sum of the percentages of the air flows generated from a CRAC unit or a compute node must equal to 1. This is guaranteed by Constraint 1. Constraint 2 guarantees that the sum of the air flows at the inlet of a CRAC unit or a compute node is equal to its air flow rate. Constraints 3 and 4 guarantee that the sum of the exit coefficients at a compute node is within the range shown in Table D.1, and that the percentages set by matrix M are satisfied. The range of the recirculation coefficients for each node is guaranteed by Constraint 5.

Appendix E

TABLE E.1. PERCENTAGE OF UNEXECUTED TASKS IN EACH TYPE FOR PROBLEM 1.

simulation #	**static power**	V_{prop}	**Technique**	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**	**T7**	**T8**
1	10	0	Ours	0	0	100	0	53	0	100	100
			Equation 22	0	0	73	0	81	0	100	100
2	10	0	Ours	0	0	0	0	0	100	100	93
			Equation 22	0	0	0	0	0	100	100	83
1	10	10	Ours	100	0	0	100	100	0	8	0
			Equation 22	100	69	0	100	100	0	0	0
2	10	10	Ours	100	0	35	45	0	0	0	88
			Equation 22	0	0	96	100	5	0	0	100
1	10	30	Ours	0	0	0	4	91	24	9	65
			Equation 22	0	100	0	0	100	16	0	81
2	10	30	Ours	82	0	89	75	0	0	0	0
			Equation 22	14	100	100	100	0	0	0	0
1	20	0	Ours	0	0	100	0	45	0	100	100
			Equation 22	0	0	73	0	81	0	100	100
2	20	0	Ours	16	0	0	0	0	95	100	60
			Equation 22	0	0	0	0	0	100	100	93
1	20	10	Ours	100	0	41	32	0	0	100	24
			Equation 22	100	69	0	100	100	0	0	0
2	20	10	Ours	0	0	0	48	0	100	0	100
			Equation 22	0	0	96	100	5	0	0	100
1	20	30	Ours	0	0	13	0	100	0	5	97
			Equation 22	0	100	0	0	100	16	0	81
2	20	30	Ours	73	0	45	100	0	0	0	51
			Equation 22	14	100	100	100	0	0	0	0
1	30	0	Ours	0	0	100	0	53	0	100	100
			Equation 22	0	0	73	0	81	0	100	100
2	30	0	Ours	0	0	0	0	0	100	100	83
			Equation 22	0	0	0	0	0	100	100	93
1	30	10	Ours	100	0	0	100	100	0	34	0
			Equation 22	100	69	0	100	100	0	0	0
2	30	10	Ours	0	0	45	19	0	100	0	100
			Equation 22	0	0	96	100	5	0	0	100
1	30	30	Ours	73	0	92	0	0	0	0	0
			Equation 22	14	100	100	0	0	0	0	0
2	30	30	Ours	0	0	41	0	100	0	5	95
			Equation 22	16	0	100	0	78	0	0	100

TABLE E.2. PERCENTAGE OF UNEXECUTED TASKS IN EACH TYPE FOR PROBLEM 2.

simulation #	static power	Vprop	Technique	T1	T2	T3	T4	T5	T6	T7	T8
1	10	0	Ours	0	0	90	0	86	62	100	93
			Equation 22	0	19	100	0	100	0	100	100
2	10	0	Ours	44	75	0	35	0	100	83	66
			Equation 22	100	12	0	0	0	100	100	100
1	10	10	Ours	100	0	56	100	100	15	67	0
			Equation 22	100	100	0	100	100	0	58	0
2	10	10	Ours	0	8	89	100	89	25	0	94
			Equation 22	17	0	100	100	100	0	0	100
1	10	30	Ours	0	100	20	100	56	0	47	100
			Equation 22	76	100	100	0	0	0	100	0
2	10	30	Ours	100	95	59	0	0	100	46	0
			Equation 22	100	92	0	100	0	0	100	0
1	20	0	Ours	0	27	90	0	100	0	100	100
			Equation 22	0	19	100	0	100	0	100	100
2	20	0	Ours	93	82	0	16	0	77	100	47
			Equation 22	100	12	0	0	0	100	100	100
1	20	10	Ours	28	0	100	100	87	2	0	100
			Equation 22	17	0	100	100	100	0	0	100
2	20	10	Ours	0	0	68	84	4	100	100	35
			Equation 22	0	0	100	100	0	100	83	0
1	20	30	Ours	0	81	28	0	97	65	0	100
			Equation 22	0	100	0	0	100	70	0	100
2	20	30	Ours	100	0	100	100	0	29	0	85
			Equation 22	100	100	100	100	18	0	0	0
1	30	0	Ours	0	0	80	35	88	28	100	100
			Equation 22	0	19	100	0	100	0	100	100
2	30	0	Ours	0	88	0	53	0	81	100	83
			Equation 22	100	12	0	0	0	100	100	100
1	30	10	Ours	100	41	0	100	100	0	94	0
			Equation 22	100	100	0	100	100	0	58	0
2	30	10	Ours	8	0	50	100	84	58	0	100
			Equation 22	17	0	100	100	100	0	0	100
1	30	30	Ours	0	60	5	0	100	91	0	100
			Equation 22	0	100	0	0	100	70	0	100
2	30	30	Ours	99	0	100	100	29	0	0	100
			Equation 22	100	100	100	100	18	0	0	0

Appendix F

TABLE F.1. TABLE OF NOTATIONS.

Notation	Description
$\boldsymbol{\alpha}$	Matrix of the cross interference coefficients
$\alpha_{i,j}$	cross interference coefficient between CRAC unit i (or compute node i – NCRAC if i is greater than NCRAC) and CRAC unit j (or compute node j – NCRAC if j is greater than NCRAC)
η_j	Number of P-states of core type j
λ_i	Arrival rate of task type i
$\pi_{j,k}$	Power consumption of core type j running in P-state k
ρ	Density of air
Φ	Power multiplier that is used to select a power constraint
AER(i, k)	Actual execution rate of tasks of type i on core k
B_j	Base power consumption of a compute node type j
$C^{\text{ECS}}_{i,j}(\text{PCORE}_k)$	ECS for a task of type i running on a core of type j as a continuous function of PCORE_k
C_L	Capacitive load
CoP(τ)	Coefficient of performance of a CRAC as a function of its outlet temperature τ
cores_j	Set of indices of cores that belong to compute node j
Cp	Specific heat capacity of air
CT_k	Type of compute node that core k belongs to (equivalently, it is the type of core k)
d_i	A value that is added to the arrival time of a task to obtain its individual deadline
DF(i, k)	Desired fraction of time that core k spends executing tasks of type i
$E(i, j)$,	Effective number of cores at compute node j that are used to execute tasks of type i
$\text{EC}_{i,j}$	Exit coefficient of compute node i for CRAC unit j
ECS(i, j, k)	Estimated computation speed of task type i on core type j running in P-state k
ER(i, k)	Desired execution rate of tasks of type i on core k
F	Vector of air flow rates $([\text{FCRAC}_1, \ldots, \text{FCRAC}_{\text{NCRAC}}, \text{FCN}_1, \ldots, \text{FCN}_{\text{NCN}}]^T)$
$f_{j,k}$	Clock frequency of a core of type j running in P-state k
FCN_i	Air flow rate at compute node i
FCRAC_i	Air flow rate at CRAC unit i
FRAC(i, j)	Fraction of compute resource (i.e., the number of cores) used at compute node j used to execute tasks of type i
HA_j	Hot-aisle to which compute node j belongs
L_j	Label of compute node j
M(i, j)	Percentage of the EC of a compute node in hot aisle i that goes to CRAC unit j
MaxEC_l	Maximum exit coefficient for a compute node with label l
MaxECS_i	maximum ECS for task type i over all core types and all P-states except the turned-off P-state
$\underline{\text{MaxRC}_l}$	Maximum RC for a compute node with label l

continue table on next page

Notation	Description
MinEC_l	Minimum exit coefficient for a compute node with label *l*
MinECS_i	minimum ECS for task type *i* over all core types and all P-states except the turned-off P-state
MinRC_l	Minimum RC for a compute node with label *l*
NCN	Number of compute nodes in the data center
NCORES	Total number of cores in the data center
NCRAC	Number of CRAC units in the data center
NT_j	Type of compute node *j*
NTYPES	Number of compute node types
$\underline{\text{PCN}}_j$	Power consumption of consumption node *j*
P_{const}	Total power constraint for Problem 1
PCORE_k	Power assigned to core *k*
$P_{\max}$	Upper bound on the maximum power consumption of the data center
$P_{\min}$	Upper bound on the minimum power consumption of the data center
PS_k	Assigned P-state of core *k*
r_i	Reward for completing a task of type *i* by its individual deadline
rand[a, b]	Uniform random variable in the interval [a, b]
RC_i	Percentage of the air flow at the inlet of compute node *i* that is recirculated from the outlet of other compute nodes
R_{const}	Reward rate constraint for Problem 2
RER(i, k)	Reassigned desired execution rate of task type *i* on core *k*
S	Number of transistor switches per clock cycle
SC	Product of S and C_L
SumECS_i	ECS obtained for a task type *i* if all cores in the data centers are used equally by every task type and all cores are running in P-state 0
T	Number of task types
$\boldsymbol{T}^{\text{in}}$	Vector of inlet air temperatures $\left(\left[\text{TCRAC}_1^{\text{in}}, \ldots, \text{TCRAC}_{\text{NCRAC}}^{\text{in}}, \text{TCN}_1^{\text{in}}, \ldots, \text{TCN}_{\text{NCN}}^{\text{in}}\right]^{\text{T}}\right)$
$\boldsymbol{T}^{\text{redline}}$	Vector of redline temperature constraints on inlet air temperatures
$\boldsymbol{T}^{\text{out}}$	Vector of outlet air temperatures $\left(\left[\text{TCRAC}_1^{\text{out}}, \ldots, \text{TCRAC}_{\text{NCRAC}}^{\text{out}}, \text{TCN}_1^{\text{out}}, \ldots, \text{TCN}_{\text{NCN}}^{\text{out}}\right]^{\text{T}}\right)$
TCN_i^{in}	Inlet air temperature at compute node *i*
$\text{TCN}_i^{\text{out}}$	Outlet air temperature at compute node *i*
$\text{TCRAC}_i^{\text{in}}$	Inlet air temperature at CRAC unit *i*
$\text{TCRAC}_i^{\text{out}}$	Outlet air temperature at CRAC unit *i*
TE_i	Task type *i* easiness
$V_{j,k}$	Supply voltage
V_{arrival}	Variation parameter used to introduce some randomness in the assigned arrival rates of task types
continue table on next page	

Notation	Description
V_{ECS}	Parameter used to obtain a variation factor to generate the ECS matrix for P-state 0
V_{prop}	Parameter used to control the variation factor so that the ECS is not exactly proportional to the clock frequency