4-2018

Adjustment of Parametric Dynamic Scheduling Heuristics for Heterogeneous Systems to Account for Heterogeneity

Thomas G. Guerin
tgg7337@rit.edu

Adjustment of Parametric Dynamic Scheduling Heuristics for Heterogeneous Systems to Account for Heterogeneity

Thomas G. Guerin

Adjustment of Parametric Dynamic Scheduling Heuristics for Heterogeneous Systems to Account for Heterogeneity

Thomas G. Guerin

April 2018

A Thesis Submitted
in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in
Computer Engineering

R·I·T | Kate Gleason
College of ENGINEERING

Department of Computer Engineering

Adjustment of Parametric Dynamic Scheduling Heuristics for Heterogeneous Systems to Account for Heterogeneity

Thomas G. Guerin

Committee Approval:

Dr. Sonia Lopez Alarcon — Date
Thesis Advisor
Department of Computer Engineering
Rochester Institute of Technology

Dr. Amlan Ganguly — Date
Department of Computer Engineering
Rochester Institute the Technology

Dr. Roy Melton — Date
Department of Computer Engineering
Rochester Institute the Technology

Acknowledgments

First and foremost, I would like to express my gratitude to my advisor Dr. Sonia Lopez Alarcon. Her patience and dedication to my success were necessary for me to succeed in my research. I would not have been able to navigate this lengthy and open-ended process without her guidance. I would also like to thank Dr. Marcin Łukowiak not only for his advisement of my research, but for the challenges, opportunities, and assistance he provided throughout my other academic pursuits. I also wish to thank Dr. Amlan Ganguly and Dr. Roy Melton for challenging me to grow in their classes, but also for serving on my thesis committee. Last, but not least, I must thank my family — especially my parents — for their support and encouragement. I have been able to rely on them from my youngest years to the day that I am writing these words.

Dedicated to my parents William George Guerin and Susan Donnelly Guerin

Abstract

Modern computing applications are becoming increasingly data-hungry and computationally expensive. This trend continues even as hardware performance constraints loom with the impending death of Moore's law. Hence, systems have become increasingly heterogeneous in the pursuits of improving performance and reducing power consumption. Such a heterogeneous system relies on a variety of different specialized processors with differing architectures, rather than processing units of a single type. Given their architectural differences, any given computation will not perform equally on all processors. As such, efficient scheduling of computations to processors is an essential design consideration.

In this thesis work, a simulation of an existing dynamic scheduling heuristic — Alternative Processor within Threshold (APT) — was used to model the execution of a variety of heterogeneous workloads and heterogeneous systems. An extended version of this scheduler (APTX) was analyzed in a similar way, as was a simplified version of the existing K-Percent Best (KPB) scheduler. Each of these schedulers has a numeric "parameter" constraining its behavior. In existing analyses, these scheduling heuristics were tested only with a small set of arbitrary values for these parameters. The goal of this research was to use a stochastic method to optimize said parameters for the minimum finishing time of any given set of computations on any given heterogeneous system. An analytical expression to estimate the ideal parameter of each scheduler was developed. Each was based on the statistical analysis of the results of a set of randomly-generated simulations.

After these expressions were developed, these optimized APT, APTX, and KPB schedulers were evaluated against three other dynamic schedulers — Minimum Execution Time (MET), Serial Scheduling (SS), and Shortest Process Next (SPN) — by running the randomly-generated simulations on all six. For the most common type of heterogeneous system, APT and APTX were found to have the earliest finish time

on average, while MET and KPB generally performed poorly. Ultimately, this research not only demonstrated the advantages of APT and APTX over other dynamic schedulers in a fair comparison, but it also demonstrated a method by which any parametric scheduler can be tuned.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Modern computing applications are increasingly data-hungry and consequently are increasingly computationally expensive [1]. Over 20% of the server market is dedicated to financial modeling, scientific computation, or data analytics. Additionally, even with these large problem sizes, many applications come with high performance requirements. As such, a naive solution is simply to add more machines to network and distributed computing clusters [1, 2]. However, this is neither cost-effective nor power-efficient, and it is therefore not a tenable long-term solution.

Conventionally-speaking, distributed computing systems have been considered to contain a large number of general-purpose CPUs [2]. Such a configuration can be appropriate for computations with a single data stream per processor. However, it was not until the early 2000s that GPUs were used as coprocessors for general purpose computing. This CPU-GPU configuration is known as General Purpose Computing on Graphics Processing Unit (GPGPU). GPUs are particularly well-suited to applications with a high degree of data parallelism, fitting the SIMD paradigm.

Field-Programmable Gate Arrays (FPGAs) also offer performance benefits for certain applications. One inherent advantage is that they scale well with Moore's law. An increase in transistor count translates to an increase in logic and memory resources. Hence, additional specialized computations can be performed in parallel in the same hardware area. Application-Specific Integrated Circuits (ASICs) offer

better performance for the applications to which they are tailored.

A system which contains more than one processor architecture — including, but not limited to CPU, GPU, FPGA, and ASIC — is known as a *heterogeneous system*. Such a system aims to provide good performance for a wide variety of applications, not just those which conform to a single architecture. A block diagram of a generic heterogeneous system is shown in Figure 1.1.

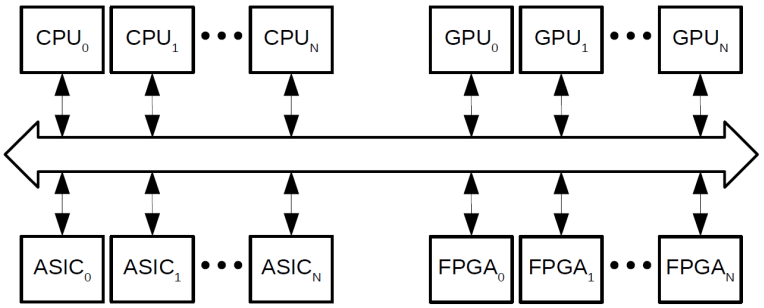

Figure 1.1: Block Diagram of a Generic Heterogeneous System

1.1 Background & Motivation

1.1.1 Limitations of Homogeneous Computing

The limitations of homogeneous computing systems are numerous and well-documented. These systems consist of one or more machines of the same type [2, 3]. While these systems have provided adequate performance for many applications in the past, they have inherent performance limitations. Any given machine will be suitable only for computations which have the same type of embedded parallelism. As a consequence, processors in a homogeneous system will spend a significant amount of time executing code for which they are poorly-suited. Programmers spend a great deal of time and effort in pursuit of making software run better on a homogeneous architecture [4]. However, the hardware architecture on which software runs imposes an inherent limit to the performance of that software.

Usually, each machine uses only one mode of parallelism [2]. For instance, GPUs fit the SIMD paradigm and perform well when executing programs with a large amount of parallel data and few divergent branches [5]. If the data are not accessed in parallel, the program executes sequentially. Divergent branches serialize execution. In either case, the parallel advantages of the GPU architecture erode.

Consider the hypothetical situation illustrated in Figure 1.2. An application has three parts each with different type of embedded parallelism: A, B, and C, respectively. Each takes 40 units of time on a general-purpose CPU. However, if this application were executed on a machine with architecture A, the first part could execute in 10 time units, and the other parts could execute in 50 time units each. While this still is an improvement overall, the processor still spends most of its time performing computations to which it is poorly-suited. In fact, two parts of the application perform slower.

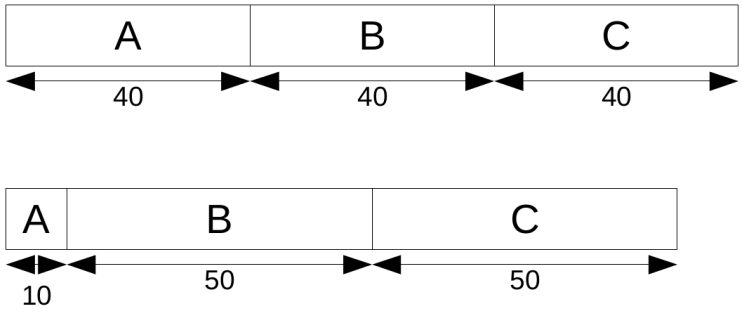

Figure 1.2: Example of Application with Multiple Types of Parallelism

1.1.2 Heterogeneous Computing

Heterogeneous computing (HC) has often been proposed as a solution to many of the limitations of homogeneous computing. Such a system has a collection of different high-performance machines connected by a high-speed network [2, 6]. Applications employing more than one type of parallelism can then be broken into segments or "tasks". Heterogeneity exists in a number of difference configurations, ranging from

data center to system on a chip (SoC).

Once again, consider the application consisting of three tasks, each with a different mode of parallelism. On a heterogeneous system, each task could be run on the machine for which it is best suited. Assuming that the communication delays resulting from dependencies are negligible, the finish time of the application could be minimized, as illustrated in Figure 1.3. Task A is still executed on its best machine, taking only 10 units of time. However, tasks B and C also are run on their respective best machines, allowing them to execute in 15 and 20 time units. In total, this application could complete in 45 time units — less than in either of the other two scenarios.

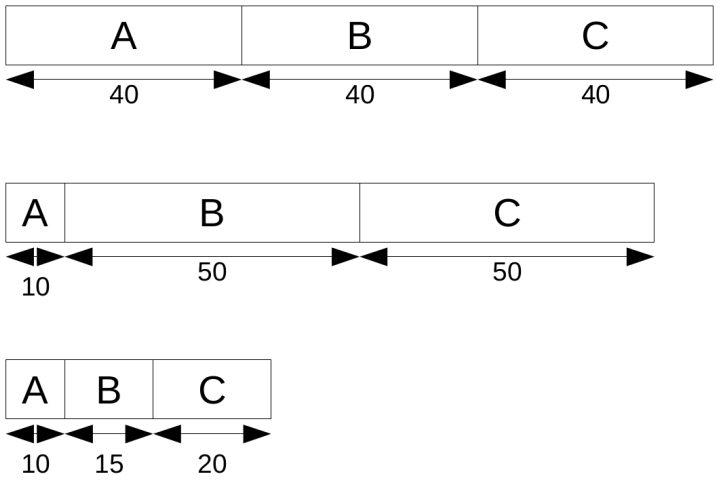

Figure 1.3: Example Application Running on a Heterogeneous System

1.1.3 Module Allocation (Mapping)

While the examples above simply illustrate the potential of HC, there are many challenges to be addressed in the development of such systems. One such challenge lies in the mapping of tasks to machines in such a way that the finishing time of the last task is minimized. This problem is sometimes known as the Module Allocation problem [7]. This is typically handled in one of two ways: *statically* or *dynamically*.

These two types of scheduling occur at different times [8, 9, 10, 11]. A *static* schedule is established before run-time. The mappings and order of the tasks are set out front. Therefore, a static scheduler requires that the entire workload is also known prior to run-time. In contrast, a *dynamic* scheduler maps tasks to machines as they become available. These policies attempt to make the best assignment given the current state of the system.

1.2 Proposed Solution

In this research, a simulator will be used to evaluate four different scheduling heuristics and produce analytical expressions to estimate the best possible parameter for each of them. Hypothetically, this number changes in accordance with the characteristics of the system. Such characteristics include measurements of machine heterogeneity, measurements of task heterogeneity, and processor count. These "features" will be used as dependent variables to the expressions for each parameter. Finally, these schedulers, with their estimated parameters, are tested against three additional scheduling heuristics.

Certain common assumptions will be carried forward from the comparison studies performed by Braun et al. and Maheswaran et al.[12, 13, 14]. First, tasks are assumed to be independent. Therefore, data transfer times are zero. Second, the running time of the heuristic is considered to be negligibly small in comparison to those of the tasks themselves. Third, estimates of the execution time of each task on each processor are assumed to be accurate. Fourth, it is assumed that there are no priorities to the tasks. Lastly, the simulation model will have only a single global task structure. Tasks are pushed to an eligible processor only as one becomes available.

Given the assumption of a single global task structure, the K-Percent Best (KPB) scheduler will be simplified somewhat for this research. The original implementation determines eligible mappings based on the execution time of the task, selects the

mapping among those which would result in the earliest completion time for that task, and then appends that task to the local task queue for the mapped processor. Assuming a global task structure ensures that mappings can't be made to a busy processor, thereby making completion time for a task mapping negligible for our purposes.

In addition, a new scheduling heuristic Alternative Processor within Threshold Extended (APTX) will be proposed and analyzed. As the name suggests, it is derived from the original APT proposal. The key difference will be the consideration of an unlimited number of processors within the specified threshold, as opposed to just one or two. The rationale is that if a threshold is suitable for the second-best processor, it would also be acceptable for others within that same threshold. This could further reduce the amount of time tasks spend waiting for assignment, by making more mappings permissible.

In order to evaluate the performance of the four schedulers (MET, APT, KPB, and APTX) in a fair manner, the simulator developed by Karia et al. will be adapted to operate on the matrix model proposed by Braun et al. [10]. Braun's methods for randomly-generating heterogeneous systems and workloads will be employed with various processor counts, task heterogeneities, machine heterogeneities, and workload sizes. Each such "experiment" will be run on each of the four schedulers in the comparison. In the case of parametric schedulers (APT, KPB, and APTX), many simulations will be run in order to search for the parameter which provides the best finish time for the workload. Next, a multivariate regression analysis will be run to determine an equation for estimating the best parameter for each of those three schedulers. Ultimately, the best scheduler and a corresponding analytical expression will be found.

1.3 Related Work

The performance benefits of alternative architectures for certain types of tasks have been demonstrated repeatedly. For instance, Fletcher et al. demonstrated the performance advantage of FPGAs over (higher-clocked) GPUs for a Bayesian interference algorithm [15]. Similarly, Hussain et al. demonstrated FPGA performance and power advantages over a software implementation of K-means microarray clustering, a biology research application [16]. Chen et al. implemented an OpenCL document filtering algorithm for an FPGA and found it to perform faster than CPU and GPU implementations [17]. Binotto et al. developed a system containing an CPU, a GPU, and an FPGA which outperformed a homogeneous CPU system for x-ray image processing [18]. Skalicky et al. developed a system containing an CPU, a GPU, and an FPGA which outperformed both a homogeneous CPU system and a homogeneous GPU system when applied to linear various linear algebra computations [19].

As for scheduling, Fernandez-Baca proved that static module allocation for minimum finishing time an NP-complete problem [7]. This necessitates the development of scheduling heuristics [6]. Numerous such heuristics, both static and dynamic, have been developed over the past several years [2, 11, 13, 20, 21, 22, 23, 24, 25]. A key challenge in the comparison of heuristics from different papers is the fact that each made different assumptions about the model of the distributed system and about the workload. Subsequent studies have attempted to compare schedulers using common sets of assumptions.

One such comparison study was performed by Braun et al., who established a matrix model for tasks on a heterogeneous system [12]. This study compared a number of static scheduling heuristics, including Opportunistic Load Balancing (OLB), static Minimum Execution Time (MET), static Minimum Completion Time (MCT), Min-min, Max-min, Duplex, a Generic Algorithm (GA), Simulated Annealing (SA),

Genetic Simulated Annealing (GSA), Tabu search, and A*.

Maheswaran et al. performed a similar comparison study for a number of different dynamic scheduling heuristics, including a dynamic variant of MET and K-Percent Best (KPB) [13]. In the dynamic MET policy, an arbitrary task is selected and assigned to the machine on which it would run the fastest if and only if that machine is available. In the KPB policy, an arbitrary task is selected. The k-percent of machines which would provide that fastest execution time for that task are considered for assignment. Then from that set, the available machine with the earliest completion time is selected for task assignment.

Karia et al. performed another dynamic scheduler comparison study and introduced a new heuristic known as Alternative Processor within Threshold (APT) [10]. APT extends the number of machines which MET considers for task assignment. An arbitrary task is considered for assignment. If available, the machine which would be fastest for that task is selected. If not, the second best machine will be selected if and only if the expected execution time is below some threshold of execution time. This threshold is defined by the value α, which serves as a ratio of the maximum permissible execution time to the best possible execution time for a task. One application was simulated with each $\alpha \in \{1.5, 2.0, 4.0, 8.0, 16.0\}$. From that set, it was found that $\alpha = 4.0$ produced the earliest finish time. Assuming one minimum on $1.0 < \alpha < \infty$, this suggests that the best α for that application lies somewhere in the range $2.0 < \alpha < 8.0$. However, this estimate is based on a limited set of workloads and a single machine configuration. The research presented in this paper looks to develop an equation for a more accurate estimation of α.

1.3.1 Motivation

The limitations of the α selection by Karia et al. form the primary motives for this research. First, only one application (workload) was tested. The ideal α may vary

depending on workload. Second, the research was performed with a unique set of assumptions, as tends to be the case with scheduler research. Third, simulations for just five values of α were performed. Of course, this is very imprecise. Hence, this research produces an equation for α based on the characteristics of the system on which APT runs, particularly heterogeneity.

Two additional concerns motivate the research being presented. First, the researched performed by Maheswaran et al. tested the KPB scheduler only with $k = 20$. Therefore, not only is KPB similar to APT in that it has a parameter, but it also has not been been tested across a wide range of k. The research presented in this thesis develops an equation to compute k as well. Second, the APT algorithm raises a question: Why only consider the best *two* machines for a task, when multiple may provide an execution time within the same multiple α of the best execution time? Hence, this research proposes and evaluates an extended version of APT with this change made.

Chapter 2

Methodology

In this chapter, the mathematical model of a heterogeneous system is described, along with all assumptions. Next, each of the eight tested scheduling policies are described. Next, the method for estimating the ideal parameter values of each scheduler is presented: α for APT, α for APTX, and k for KPB.

A stochastic method was necessary for the development of for the parameter of each scheduler because. There is no way to construct a system of one or more equations that can be mathematically solved for α or k.

2.1 Simulation Model, Assumptions, and Definitions

This research used the same model as [12, 13, 14]. A heterogeneous system consists of a set of μ distinct machines $M = \{m_1, m_2, ..., m_\mu\}$. The workload of tasks to run on the system consists of τ tasks $T = \{t_1, t_2, ..., t_\tau\}$. It is assumed that the execution time of each task on each machine is known, such that $e_{i,j}$ represents the execution time of task i on machine j. These values populate a matrix of size $\tau \times \mu$ known as the Expected Time to Compute (ETC) matrix. As such, row i of the matrix contains the estimated execution times of a given task $t_i, 1 < i < \tau$ on each machine in M. Similarly, column j of the matrix contains the estimated execution time of each task in T on a given machine $m_j, 1 < j < \mu$. Additionally, let $A \subseteq M$ be the available (i.e., not busy) machines in the system.

An ETC matrix can be classified into one of three types of *consistency* [12, 14]. Braun et al. define each type of consistency as follows:

- An ETC matrix is said to be *consistent* if, in every possible pair of machines, the execution times of one are all greater than or equal to those of the other.
- A *partially-consistent* ETC matrix contains a consistent submatrix.
- An *inconsistent* ETC matrix has no consistent submatrix. In other words, it is any ETC matrix which is neither consistent nor partially-consistent.

Assumptions will be carried forward from the comparison studies performed in [12, 13, 14]. First, tasks are assumed to be independent. Therefore, data transfer times are zero. Second, the running time of the heuristic is considered to be negligibly small in comparison to those of the tasks themselves. Third, estimates of the execution time of each task on each processor are assumed to be accurate. Fourth, it is assumed that there are neither priorities nor deadlines to the tasks. Lastly, the simulation model will have only a single global task structure. Tasks are pushed to an eligible processor only as one becomes available.

In this research, given the assumption of a single global task structure, the K-Percent Best scheduler is simplified somewhat. In this policy, the "k percent" of processors which would provide the best execution time for a task are considered for mapping. Then, in the original implementation, the mapping among those which would result in the earliest completion time for that task is selected. The task in question is then appended to the local task queue of that processor. Assuming a global task structure ensures that mappings can't be made to a busy processor, thereby making completion time for a task mapping meaningless.

With the help of the work by Karia et al., Java was used to develop a simulator to model the execution of heterogeneous applications on heterogeneous systems [10]. Each simulation run requires four pieces of information. The first among those is

the ETC matrix. This defines both the machines comprising the system and the workload of tasks to run. Second, the bandwidth for inter-machine communications is required. Third, the structure of the data flow graph (DFG) of task dependencies is required [26]. While it is true that this research does not consider transfer times or dependencies, the simulator is able to handle them nonetheless. In this case, independent tasks mean that the nodes of the DFG are not connected. Fourth, the simulation requires a scheduling policy to be specified, including any parameters used by that policy.

Naturally, the simulation time starts at 0.0 ms. The simulation procedure is iterative. Iterations are performed until all tasks in the DFG are completed. In each iteration, if there are any idle machines (those which do not have an assigned task with time remaining), the scheduler attempts assign a task to a machine. The criteria for a successful assignment vary with each scheduler. Next, the least amount of time until a busy processor will finish a task is calculated. This "step time" is deducted from the compute time remaining on each busy processor, and the simulation time is incremented by the same amount. This procedure is summarized in Algorithm 1.

Algorithm 1 Simulation Procedure

```
1:  function SIMULATE(DFG, system)
2:      simTime ← 0
3:      while uncompleted tasks ∈ DFG do
4:          if A ≠ ∅ then
5:              AssignNext
6:          timeStep ← time to next task completion
7:          for Processor p ∈ system do
8:              if p is running a task then
9:                  p.timeRemaining ← p.timeRemaining − timeStep
10:         simTime ← simTime + timeStep
11:     return simTime
```

2.2 Policies

In this research, six different dynamic scheduling policies are described, modeled, and analyzed. The first — Minimum Execution Time (MET) — is the simplest and serves as the basis of comparison for each of the other schedulers. Three policies each take a numeric parameter before execution. This parameter imposes some constraint on the machines which are considered for any given task. These three policies are known as Alternative Processor within Threshold (APT), K-Percent best (KPB), and Alternative Processor within Threshold Extended (APTX). Two additional dynamic policies — Serial Scheduling and Shortest Process Next — are also considered.

2.2.1 Minimum Execution Time (MET)

Dynamic Minimum Execution Time, sometimes known as Best Only, is the simplest scheduling heuristic in this research. This scheduler attempts to assign an arbitrary uncompleted task to the machine on which it would execute in the least amount of time [10, 13]. If that processor is busy, then the policy waits until it becomes available to perform the mapping. This procedure, as implemented in the simulator, is summarized in Algorithm 2.

Algorithm 2 MET Assignment Procedure

```
1: function AssignNext
2:     for uncompleted unassigned task t_i ∈ DFG do
3:         Machine m_best ← m_k ∈ M : ETC(i,k) = Min{ETC(i)}
4:         if m_best ⊆ A then
5:             Assign t_i to m_best
6:             m_best.timeRemaining ← ETC(i, best)
7:             return
```

The major downside to MET is the fact that it can lead to severe load imbalance across the machines. A backlog of tasks may end up waiting for a small number of machines to become available, especially if the ETC matrix is partially-consistent or

consistent. The major advantages are the simplicity (i.e., low cost) of the decision and the fact that a task will never run on a machine to which it is poorly-suited.

2.2.2 Serial Scheduling (SS)

Serial Scheduling takes a statistical approach to the assignment problem [27]. This policy considers the distribution of execution times among available processors for each task. That is to say, every task is given consideration for assignment priority. For each task, the mean and standard deviation of execution times on available processors is computed. The scheduler chooses the task t_i with the highest such standard deviation and assigns it to machine $m_j \in A$ with the lowest execution time.

One advantage to SS is that tasks never wait for assignment as one or more machines sit idle. Load balancing is generally good as a consequence. Second, the tasks with the highest heterogeneity (represented by the standard deviation) are prioritized, as it would have the relatively-highest penalty for poor assignment. However, this does not ensure that poor assignments are impossible. Processors will always be given a task, even if that assignment is poor.

2.2.3 Shortest Process Next (SPN)

The Shortest Process Next policy is also very simple. Among uncompleted tasks and available processors, the task-machine mapping with the lowest overall execution time is performed. To illustrate this, suppose a system represented by the ETC matrix in Figure 2.1. Let execution begin at $time = 0$. First, task t_2 would be mapped to machine m_2. Next, task t_1 would be mapped to machine m_1. Next, task t_4 would be mapped to the only remaining machine m_3. At this point, no machines remain, so $task_2$ runs to completion at $time = 10$. At this point, only machine m_2 is available, and only task t_3 is uncompleted, so t_3 is mapped to m_2. This entire flow is represented visually in Figure 2.2.

$$\begin{bmatrix} 10 & 15 & 25 \\ 25 & 5 & 30 \\ 20 & 15 & 55 \\ 30 & 35 & 45 \end{bmatrix}$$

Figure 2.1: SPN Example Matrix

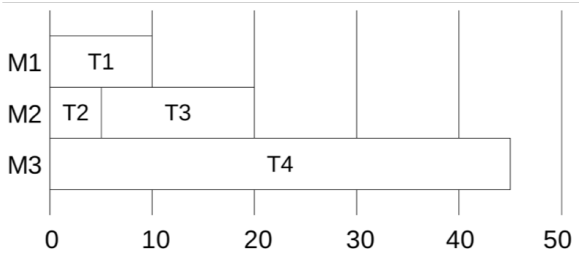

Figure 2.2: Example Application Running on SPN

Two advantages of this policy are that machines are kept busy and load balancing is good. A downside to this policy is that neither task nor machine heterogeneity is considered.

2.2.4 Alternative Processor within Treshold (APT)

Alternative Processor within Threshold is a policy which adds flexibility to the MET policy by considering more than one machine for each task in the event that the best machine is busy [10]. This flexibility can even be tuned via a numeric parameter $\alpha \geq 1$, which serves as a ratio of the maximum permissible execution time to the best possible execution time for a task assignment. The effect is that α limits the situations wherein a task is assigned to a machine other than its best. At the same time, the option to assign to an alternate machine is open in certain situations where the best machine is busy.

Processor selection in APT performs as follows. An arbitrary task is considered for assignment. Let t_{min} be the lowest execution time possible for that task and m_{best} be

the processor that would provide it. In the ideal scenario, m_{best} is already available, so APT maps the task to it. However, if that processor is busy, the second-best machine m_{alt} for the task is considered. If the expected execution time for the second-best machine $t_{alt} \leq \alpha \times t_{min}$, then the task is instead mapped to that machine. By this mechanism, tasks are kept from waiting too long for one machine, and load balancing is improved, while poor alternative machines are also avoided. The idea is that the hit to execution time can be acceptable if it is small relative to the waiting time.

The APT processor selection procedure, as implemented in the simulator, is summarized in Algorithm 3.

Algorithm 3 APT Assignment Procedure

```
1: function AssignNext(α)
2:     for uncompleted unassigned task t_i ∈ DFG do
3:         time_best ← Min{ETC(i)}
4:         Machine m_best ← m_k ∈ M : ETC(i,k) = time_best
5:         if m_best ⊆ A then
6:             Assign t_i to m_best
7:             m_best.timeRemaining ← ETC(i, best)
8:             return
9:         else
10:            time_max ← time_best × α
11:            Machine m_alt ← machine with the second best execution time for t_i.
12:            if m_alt ∈ A then
13:                Assign t_i to m_alt
14:                m_alt.timeRemaining ← ETC(i, alt)
15:                return
```

To illustrate APT, consider the ETC matrix shown in Figure 2.3. Suppose that tasks are mapped in order when possible. Let $\alpha = 2.0$. Start at $time = 0$. First, task t_1 is mapped to machine m_2. Task t_2 is then skipped for the time being, as the best machine m_2 is already occupied and the second-best machine takes more than $\alpha = 2.0$ times that of the best. Task t_3 is mapped to its best machine m_1, which is available. Task t_4 is mapped to its best second-best machine m_3. This is allowed because $30 \leq 20 \times \alpha$. Task t_5 is not mapped initially, as its two best machines are

busy. With no more mappings possible, time advances to $time = 10$, at which point machine m_2 finishes task t_1. Task t_2 is assigned to it, since it is the best machine for that task. Finally, once machine m_1 becomes available at $time = 20$, task t_5 is to it. The entire application finishes at $time = 40$. This entire flow is represented visually in Figure 2.4.

$$\begin{bmatrix} 15 & 10 & 40 & 45 \\ 35 & 15 & 45 & 50 \\ 15 & 25 & 20 & 35 \\ 20 & 35 & 30 & 40 \\ 20 & 50 & 30 & 35 \end{bmatrix}$$

Figure 2.3: APT Example Matrix

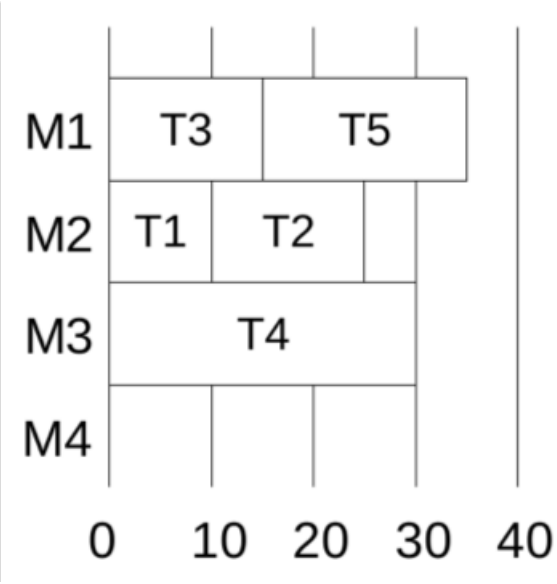

Figure 2.4: Example Application Running on APT

A contribution of this work is to develop an analytical expression to estimate the ideal value of α based on characteristics of the ETC matrix. Karia et al. did explore this to a limited extent. However, only one workload was simulated. Moreover, it was only simulated with each $\alpha \in \{1.5, 2.0, 4.0, 8.0, 16.0\}$. The research presented in

this paper simulates a vastly larger set of applications, systems, and values of α in order to determine how ETC matrix characteristics can be used to estimate the best value of α for any ETC matrix.

The entirety of this analysis consists of three main steps: random experiment generation, tests of many values of α with each experiment, and a statistical analysis of the relationship of the system (experiment) characteristics to the values of α which finished each experiment most quickly. This analysis produces an expression to estimate the best α based on the aforementioned system characteristics.

Furthermore, the APT algorithm raises a question: Why consider only the best *two* machines for a task, when multiple may provide an execution time within the same threshold the best execution time? Another contribution of this research is to determine whether or not allowing an unlimited number of processors within a threshold produces better performance than with standard APT.

2.2.5 Alternative Processor within Threshold Extended (APTX)

APTX aims to address the aforementioned possible oversight of APT. It uses α in the same way, but instead of limiting the machine selection to just the best two machines, the entire subset of machines with execution times under the threshold are considered. For instance, consider a task t_i with execution times $e_i = \{10, 15, 25, 40, 50\}$ on machines $M = \{m_1, m_2, ..., m_5\}$. Suppose that $\alpha = 3.0$ and machines m_1 and m_2 are busy. At this point, APT would wait to assign t_i to a processor, in spite of the fact that m_3 would also execute within $\alpha \times e_1$. APTX on the other hand, would assign t_i to m_3. This added flexibility should serve to further reduce the time that tasks spend waiting for assignment to a machine, without violating the threshold.

To illustrate APTX, again consider the ETC matrix shown in Figure 2.3. Suppose that tasks are mapped in order when possible. Let $\alpha = 2.0$. Start at $time = 0$. First, task t_1 is mapped to machine m_2. Task t_2 is then skipped for the time being, as the

best machine m_2 is already occupied and the second-best machine takes more than $\alpha = 2.0$ times that of the best. Task t_3 is not mapped to its best machine m_1, which is busy. Instead it is mapped to machine m_3, which is the second-best machine and is available. Task t_4 is mapped to its best machine m_1. The first and second-best machines for t_5 are busy, but the third-best machine is still under the threshold set by $\alpha = 2$, so it is also mapped initially. With no more mappings possible, time advances to $time = 10$, at which point machine m_2 finishes task t_1 and becomes available for task t_2 to be mapped to it. The entire application finishes at $time = 35$, sooner than the APT example did. This entire flow is represented visually in Figure 2.5.

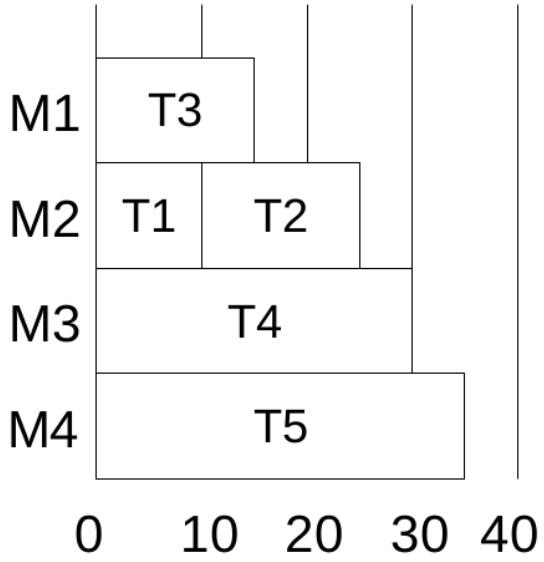

Figure 2.5: Example Application Running on APTX

This work contributes the proposal and evaluation of this scheduler. Part of this evaluation involves the determination of a good value of α, which can be tuned in accordance with the characteristics of the system on which it runs. This research attempts to develop an equation for α based on those characteristics. A stochastic approach was taken to solve this. It consists of three steps: random experiment generation, tests of many values of α with each experiment, and a statistical analysis of the relationship of the system (experiment) characteristics to the values of α

which finished each experiment most quickly. This analysis produces an expression to estimate the best α based on the aforementioned system characteristics.

2.2.6 K-Percent Best (KPB)

K-Percent Best, like APT, has behavior constrained by a numeric parameter (this time k) [13]. Again, this parameter can be tuned in accordance with characteristics of the heterogeneous system on which it runs. For each task, KPB considers a subset of the machines in M. This subset consists of the $\frac{k \times |M|}{100}$ fastest machines in the system for that task. From that set, the machine with the lowest completion time is chosen.

As mentioned, the simulation model in this research does not include individual task queues on each machine and therefore completion time is meaningless. However, the original KPB heuristic was modified to fit the simulation model and work in a fashion more similar to APT. The selection of the $\frac{k \times |M|}{100}$ fastest machines remains the same. In contrast, the available machine in that subset with the lowest *execution* time is selected. If no machine in the subset is available, no assignment occurs at that time. The task waits for one to become available. Note that k must be set sufficiently high that $\frac{k \times |M|}{100} \geq 1$ and must not exceed 100.

One additional concern motivates the research being presented. The researched performed by Maheswaran et al. tested only the KPB scheduler with $k = 20$. Therefore, not only is KPB similar to APT and APTX in that it has a parameter, but it also has not been been tested across a wide range of k. Hence, this research attempts to develop an equation for k. A stochastic approach was taken to solve this problem. It consists of three steps: random experiment generation, tests of many values of k with each experiment, and a statistical analysis of the relationship of the system (experiment) characteristics to the values of k which finished each experiment most quickly. This analysis produces an expression to estimate the best k based on the aforementioned system characteristics.

2.3 Random Experiment Generation

An "experiment" consists simply of an ETC matrix that defines the machines in a system and the workload. In this research, it was performed in a way similar to [12]. Four numbers define the characteristics used to generate each matrix. First, let τ be the number of tasks (i.e., the row count of the matrix). Second, let μ be the number of machines in the system (i.e., the column count of the matrix). Third, let ϕ_b define the task heterogeneity — the amount of variation among execution times for a task. $\phi_b = 100$ is considered low task heterogeneity while $\phi_b = 3000$ is considered high. Fourth, let ϕ_r define the machine heterogeneity — the amount of variation among execution times for a processor. $\phi_r = 10$ is considered low machine heterogeneity while $\phi_r = 1000$ is considered high.

To generate an ETC matrix (sized $\tau \times \mu$), a $\tau \times 1$ baseline column vector B was generated. It was then populated with floating-point values from a uniform random variable $X_b = U(1, \phi_b)$. Next, the rows of the matrix were generated. Each element $ETC(i, j)$ of the matrix was set to the product of B_i and another uniform random variable $X_r = U(1, \phi_r)$. Hence, $B_i \leq ETC(i, j) < B_i \times \phi_r \forall i \in [1, \tau], j \in [1, \mu]$. After generating each row, if the matrix was specified to be consistent, the values of that row were sorted.

An example of a consistent ETC matrix with six tasks, six machines, low task heterogeneity, and low machine heterogeneity (LoLo) is given in Figure 2.6. An example of an inconsistent ETC matrix with six tasks, four machines, high task heterogeneity and high machine heterogeneity (HiHi) is given in Figure 2.7.

$$\begin{bmatrix} 56.3 & 66.8 & 82.6 & 99.9 & 112.2 & 123.4 \\ 36.2 & 100.3 & 103.2 & 153.2 & 155.2 & 165.8 \\ 100.8 & 288.0 & 293.9 & 356.2 & 391.3 & 418.0 \\ 48.8 & 62.9 & 155.0 & 189.1 & 207.8 & 226.6 \\ 212.3 & 291.9 & 404.8 & 455.3 & 516.2 & 600.6 \\ 162.1 & 163.9 & 171.5 & 225.0 & 291.6 & 365.5 \end{bmatrix}$$

Figure 2.6: Consistent LoLo ETC Matrix Example

$$\begin{bmatrix} 344363.3 & 56304.0 & 373420.1 & 318912.4 \\ 1067733.3 & 1769877.1 & 79736.6 & 1243421.9 \\ 980118.2 & 1640319.5 & 705306.7 & 1410030.9 \\ 162484.9 & 1027283.3 & 853241.4 & 942676.9 \\ 96943.1 & 42296.2 & 80775.2 & 136384.5 \\ 1293686.8 & 1660723.6 & 1987079.0 & 2100383.4 \end{bmatrix}$$

Figure 2.7: Consistent HiHi ETC Matrix Example

For the purposes of this research, a consistent matrix and an inconsistent matrix were generated for every combination of the following parameters:

$$\tau = 256, 512, 1024, 2048$$

$$\mu = 4, 8, 12, 16, 20$$

$$\phi_b = 100, 150, 200, ..., 3000$$

$$\phi_r = 10, 100, 1000$$

This resulted in a set of 3540 consistent experiments and 3540 inconsistent experiments covering a wide variety of task counts, machine counts, task heterogeneities, and machine heterogeneities. The purpose of having such a varied test set is to determine which characteristics influence what an ideal parameter would be for APT, KPB, and APTX.

Chapter 3

Experimental Results & Analysis

Two phases of simulations comprise the complete analysis performed in this research. The first part involves simulating each experiment with a wide range of parameter values for APT, APTX, and KPB. The values resulting in the lowest finish times are recorded and used to develop equations for estimating each parameter. This leads into the second phase of simulations, which is performed to evaluate the estimations of each parameter and compare the six described schedulers.

Two separate simulations and equations were developed for each scheduler — one for consistent ETC matrices and one for inconsistent ETC matrices. Recall that in a consistent matrix, each machine is strictly better or worse than each other machine.

3.1 Generation of Parameter Expressions

Next, the problem of actually determining the best parameter for each experiment was addressed. This was done by simulating each experiment with a wide range of parameter values at high precision. Prior to the simulations of each experiment, various measurements of task and machine heterogeneity, and the processor count were recorded. The complete list of "features" is given in Table 3.1.

The finishing time of each simulation, scheduler, and parameter value was recorded. After running all simulations, the parameter which gave the earliest finish time was identified. In the case of multiple values producing the earliest finish time,

Table 3.1: Features Measured from Each Experiment

	No.	Name	Description
Machine Heterogeneity	1	minTaskRange	least difference between greatest and least execution time of any task
	2	maxTaskRange	greatest difference between greatest and least execution time of any task
	3	meanTaskRange	mean difference between greatest and least execution time of all tasks
	4	minTaskRatio	least ratio of greatest to least execution time of any task
	5	maxTaskRatio	greatest ratio of greatest to least execution time of any task
	6	meanTaskRatio	mean ratio of greatest to least execution time of all tasks
	7	minTaskStd	least standard deviation of execution times of any task
	8	maxTaskStd	greatest standard deviation of execution times of any task
	9	meanTaskStd	mean standard deviation of execution times of all tasks
	10	maxTaskBestRatio	greatest ratio of second-least to least execution time of any task
	11	minTaskBestRatio	least ratio of second-least to least execution time of any task
	12	meanTaskBestRatio	mean ratio of second-least to least execution time of all tasks
	13	taskMeanExtremaRange	range of the mean worst execution time of each task to the mean best execution time of each task
	14	taskMeanExtremaRatio	ratio of the mean worst execution time of each task to the mean best execution time of each task
	15	taskMeanBestRatio	ratio of the mean best execution time to the mean second-best execution time
Task Heterogeneity	16	procMeanRange	difference between the greatest mean execution time of any processor and the lowest
	17	procMeanRatio	ratio of the greatest mean execution time of any processor and the lowest
	18	procMeanStd	standard deviation of processor mean execution times
Processor Count	19	procCount	number of processors (i.e., machines) in the system

the lowest such value was used.

Next, relationships between the aforementioned features of each experiment and the corresponding parameter value (α or k) were sought. The correlations of linear relationships (x vs. y), exponential relationships (x vs. $log(y)$), logarithmic relationships ($log(x)$ vs. y), and power relationships ($log(x)$ vs. $log(y)$ were all tested for each feature. The best relationship of each feature was then used to form a term of an independent variable in a multivariate regression analysis if that relationship had an $r^2 \geq 0.25$.

3.1.1 Search for Best APT Parameter

The 5430 consistent experiments were run first, with each $\alpha = 1.0, 1.1, 1.2, ..., 8.0$, based on the assumption that it would never be a good decision to assign a task to a machine eight times slower than the fastest one. The simulated value of α which resulted in the earliest finish time was recorded for each experiment. In the case of a tie, the least α was used. Note that the best value of α was never observed to be greater than 7.4, further reinforcing the assumption that $\alpha > 8.0$ would never be ideal. Next, Octave was used to compute the correlation between each feature and α for potential linear, exponential, logarithmic, and power relationships. Those with $r^2 >= 0.25$ were recorded. Any smaller correlation was not considered to be meaningful. Listing 3.1 shows the text output of the script performing this function.

Listing 3.1: APT Consistent Correlations

```
feat 4,lin: r2=0.777182
feat 4,exp: r2=0.765742
feat 4,log: r2=0.819243
feat 4,pow: r2=0.852641
feat 6,lin: r2=0.421435
feat 6,exp: r2=0.378579
feat 6,log: r2=0.403876
feat 6,pow: r2=0.382522
feat14,lin: r2=0.806543
feat14,exp: r2=0.747996
feat14,log: r2=0.821915
feat14,pow: r2=0.829146
feat17,lin: r2=0.806543
feat17,exp: r2=0.747996
feat17,log: r2=0.821915
feat17,pow: r2=0.829146
feat19,lin: r2=0.835311
feat19,exp: r2=0.861174
feat19,log: r2=0.812226
feat19,pow: r2=0.897939
```

Each line of Listing 3.1 contains a feature number (corresponding to Table 3.1) followed by the type of relationship of the data — (lin)ear, (exp)onential, (log)arithmic, or (pow)er — followed by a coefficient of determination (the square of correlation r_2). This text was manually inspected as follows.

Only one of each type of measurement was kept. For instance, *minTaskRatio*, *meanTaskRatio*, and *taskMeanExtremaRatio* are all fundamentally measurements of machine heterogeneity. Since *minTaskRatio* gave the highest correlation of them all in a power relationship, it was used as the sole measurement of machine heterogene-

ity. This relationship is plotted in Figure 3.1. Similarly, *procMeanRatio* was the most useful measurement of task heterogeneity when in a power relationship. This is plotted in Figure 3.2. Likewise, *procCount* best indicated the correct value of α in a power relationship. This is plotted in Figure 3.3.

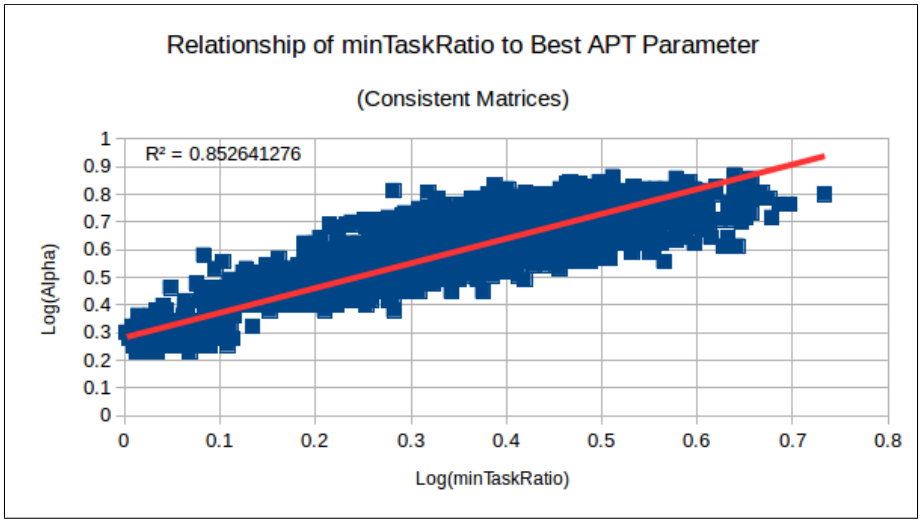

Figure 3.1: Relationship of *minTaskRatio* to Best α for APT, Consistent Systems

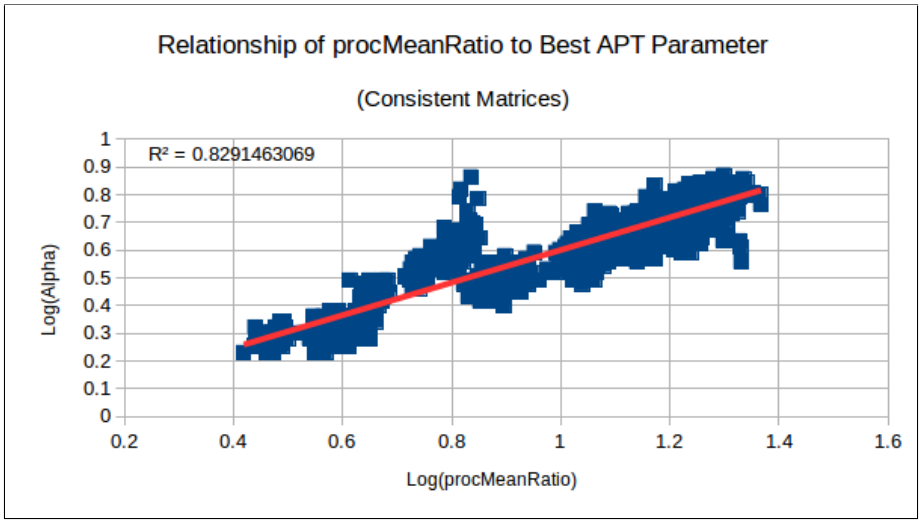

Figure 3.2: Relationship of *procMeanRatio* to Best α for APT, Consistent Systems

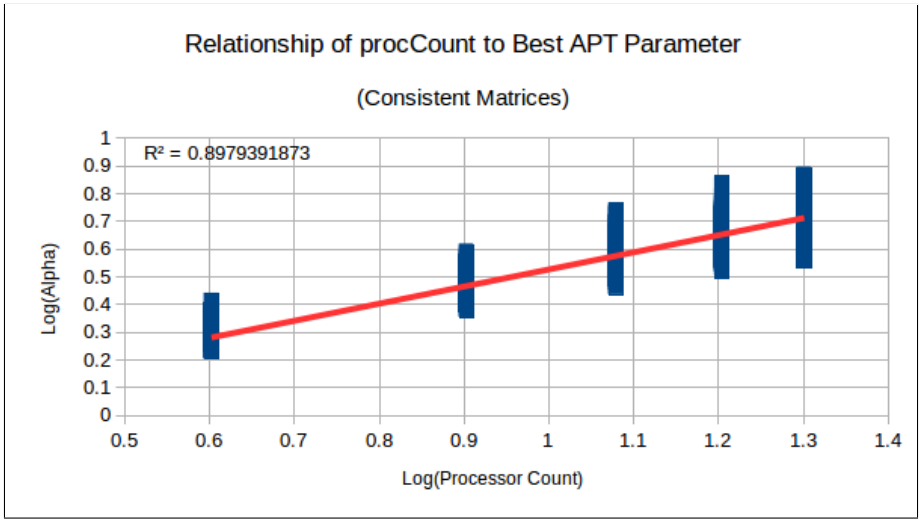

Figure 3.3: Relationship of *procCount* to Best α for APT, Consistent Systems

From these data, an equation to estimate α was developed. The equation was modeled on the aforementioned predictor-α relationships. This model is shown in (3.1). It consists of a constant and a sum of three powers. Let x_4 represent *minTaskRatio*. Let x_{17} represent *procMeanRatio*. Let x_{19} represent *procCount*).

$$\alpha = b_1 + b_2 x_4^{b_3} + b_4 x_{17}^{b_5} + b_6 x_{19}^{b_7} \tag{3.1}$$

This model and the data were run through the Matlab fitnlm (fit non-linear model) function in order to find good values for $b_i, i \in \{1, 2, ..., 7\}$. The resulting regression function is shown in (3.2). The coefficient of determination is $r^2 = 0.945$, indicating a very good fit between the equation and the observed data.

$$\alpha = -0.46606 + 1.0713 x_4^{0.0056433} + 0.17125 x_{17}^{0.90338} + 0.26261 x_{19}^{0.78846} \tag{3.2}$$

This function was integrated into the simulator to provide an automatically computed and precise value of α to use with APT and a consistent ETC matrix. However, further analysis reveals the most significant predictors of α. For instance, the power of

x_4 is very small (0.0056433). It essentially reduces to one (as a power zero does), even with the largest possible value in this research $\phi_r = 1000$. Hence, the entire x_4 term reduces to a constant. Performing this simplification and approximating the other coefficients and powers results in the more simple equation given in (3.3). Again, let x_{17} represent *procMeanRatio* and let x_{19} represent *procCount*).

$$\alpha = 0.61 + 0.17x_{17}^{0.90} + 0.26x_{19}^{0.79} \tag{3.3}$$

Hence, the most meaningful predictor of α for APT on a consistent system is processor count, as evidenced by the high coefficient. This is as one would expect for a consistent system. Each task "ranks" machines in the exact same way. Therefore, in order to keep up load balancing, α needs to allow two machines for some of the tasks and one machine for those which would take too severe of a performance hit. Naturally, in order to choose such a value of α, the "typical difference" between execution times of a task is also needed. Fundamentally, this is the meaning of task heterogeneity. Hence, the second-best predictor of α is task heterogeneity. In addition, each of these predictors diminishes in its returns as it increases. Adding one more machine to a large consistent system will not offer much of a benefit. Similarly, if task heterogeneity is high, adding just a bit more won't drastically change the ideal threshold.

Next, the 5430 inconsistent experiments were run. The same values of α were used, and the same criteria for identifying the best value of α were used. Again, Octave was used to compute the correlation between each feature and α for potential linear, exponential, logarithmic, and power relationships. Those with $r^2 >= 0.25$ were recorded. Any smaller correlation was not considered to be meaningful. Listing 3.2 shows the text output of the script performing this function. This text was inspected in the same way as for consistent APT experiments.

Listing 3.2: APT Inconsistent Correlations

```
feat 4,lin: r2=0.282954
feat 4,exp: r2=0.289308
feat 4,log: r2=0.279616
feat 4,pow: r2=0.291123
feat14,lin: r2=0.275936
feat14,exp: r2=0.284325
feat14,log: r2=0.283893
feat14,pow: r2=0.297031
feat17,lin: r2=0.313336
feat17,exp: r2=0.312759
feat17,log: r2=0.315290
feat17,pow: r2=0.315411
```

Only one of each type of measurement was kept. *taskMeanExtremaRatio* gave the highest correlation between machine heterogeneity and α. This relationship is plotted in Figure 3.4. Similarly, *procMeanRatio* was the most useful measurement of task heterogeneity when in a power relationship. This is plotted in Figure 3.5. Task heterogeneity was determined to be a poor indicator of the best α when running APT on an inconsistent system.

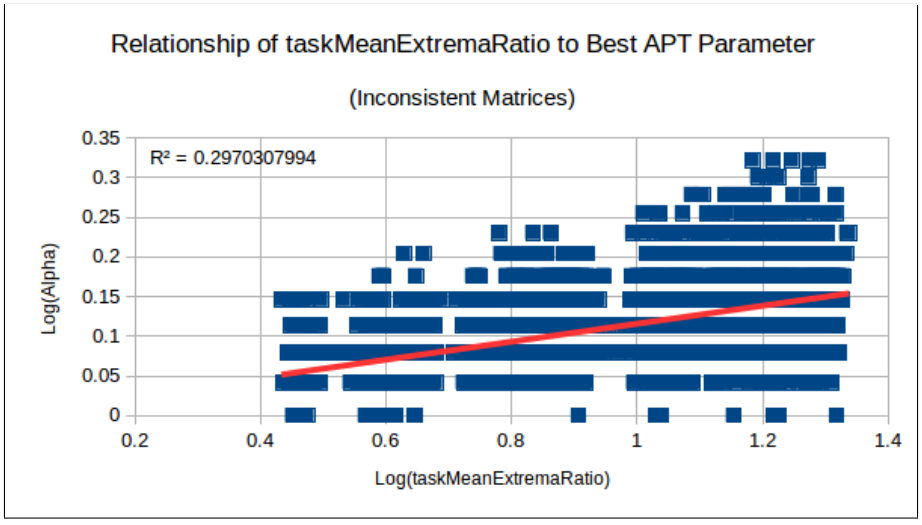

Figure 3.4: Relationship of *taskMeanExtremaRatio* to Best α for APT

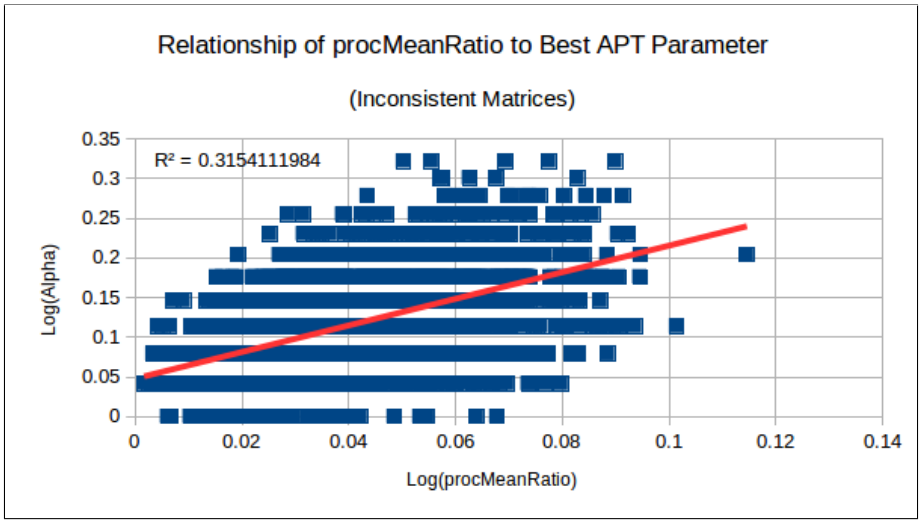

Figure 3.5: Relationship of *procMeanRatio* to Best α for APT

From these data, an equation to estimate α was developed. The equation was modeled on the aforementioned predictor-α relationships. This model is shown in (3.4). It is the sum of a constant and and two powers. Let x_{14} represent *taskMeanExtremaRatio* and let x_{17} represent *procMeanRatio*.

$$\alpha = b_1 + b_2 x_{14}^{b_3} + b_4 x_{17}^{b_5} \tag{3.4}$$

This model and the data were run through the Matlab fitnlm (fit non-linear model)

function in order to find good values for $b_i, i \in \{1, 2, ..., 5\}$. The resulting regression function is shown in (3.5). The coefficient of determination is $r^2 = 0.431$, which is much weaker than that for APT and consistent matrices.

$$\alpha = -52.731 + 0.05632x_{14}^{0.55945} + 53.694x_{17}^{0.031269} \tag{3.5}$$

This function was integrated into the simulator to provide an automatically computed and precise value of α to use with APT and an inconsistent ETC matrix. However, further analysis reveals the true predictors of α. For instance, the value (typically not more than about 1.3) and power (0.031269) of x_{17} are both so small that the whole term essentially reduces to a constant. Performing this simplification and approximating the other coefficients and powers results in the more-simple equation given in (3.6). Again, let x_{14} represent *taskMeanExtremaRatio* and let x_{17} represent *procMeanRatio*.

$$\alpha = 0.96 + 0.06x_{14}^{0.56} \tag{3.6}$$

Hence, the only significant predictor of α for APT on an inconsistent system is machine heterogeneity. Unlike in a consistent system, the same machines are not the best for every task. Hence, the purpose of α has less to do with setting the number of eligible machines, so the processor count and task heterogeneity become poor predictors.

3.1.2 Search for Best APTX Parameter

The same 5430 consistent experiments from earlier were run again, this time with APTX instead of APT. Again, each experiment was simulated with each value $\alpha \in \{1.0, 1.1, 1.2, ..., 8.0\}$. The assumption was upheld that it would never be a good decision to assign a task to a machine eight times slower than the fastest one for that

task. The same criteria for selecting the "best" value of α for an experiment were used. The lowest value of α to result in the earliest finish time of each experiment was used. Again, Octave was used to compute the correlation between each feature and α for potential linear, exponential, logarithmic, and power relationships. Those with $r^2 >= 0.25$ were recorded. Any smaller correlation was not considered to be meaningful. Listing 3.3 shows the text output of the script performing this function. This text was inspected in the same way as for APT experiments.

Listing 3.3: APTX Consistent Correlations

```
feat 4,lin: r2=0.776732
feat 4,exp: r2=0.765430
feat 4,log: r2=0.818943
feat 4,pow: r2=0.852392
feat 6,lin: r2=0.422933
feat 6,exp: r2=0.379691
feat 6,log: r2=0.404624
feat 6,pow: r2=0.383108
feat14,lin: r2=0.807905
feat14,exp: r2=0.749044
feat14,log: r2=0.822695
feat14,pow: r2=0.829768
feat17,lin: r2=0.807905
feat17,exp: r2=0.749044
feat17,log: r2=0.822695
feat17,pow: r2=0.829768
feat19,lin: r2=0.835763
feat19,exp: r2=0.861510
feat19,log: r2=0.812433
feat19,pow: r2=0.898070
```

Only one of each type of measurement was kept. In the case of consistent sys-

tems running APTX, the measurements were the same as those of consistent systems running APT. For machine heterogeneity, *minTaskRatio* gave the highest correlation in a power relationship. This relationship is plotted in Figure 3.6. Similarly, *procMeanRatio* was the most useful measurement of task heterogeneity when in a power relationship. This is plotted in Figure 3.7. Likewise, *procCount* best indicated the correct value of α in a power relationship. This is plotted in Figure 3.8.

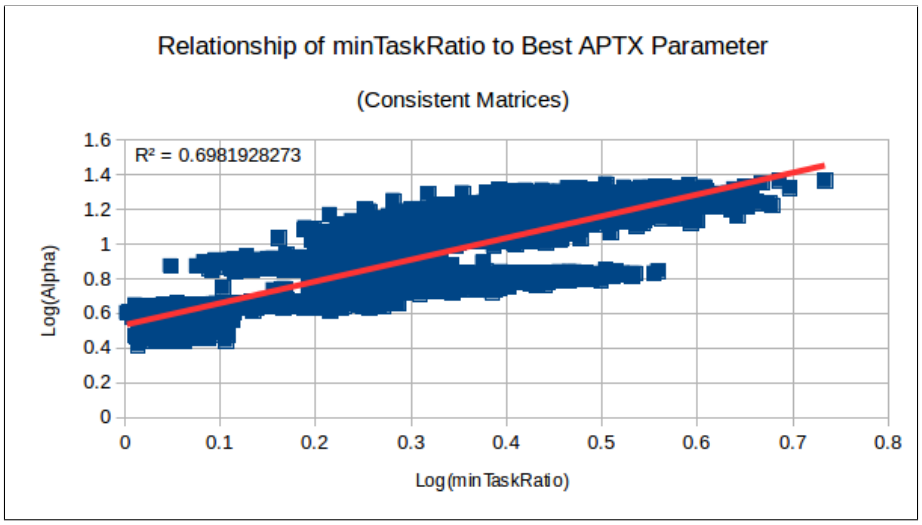

Figure 3.6: Relationship of *minTaskRatio* to Best α for APTX, Consistent Systems

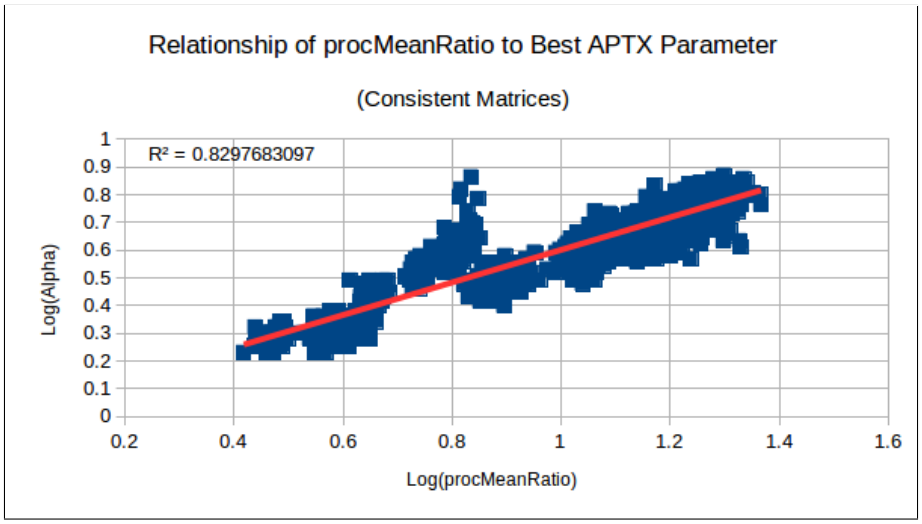

Figure 3.7: Relationship of *procMeanRatio* to Best α for APTX, Consistent Systems

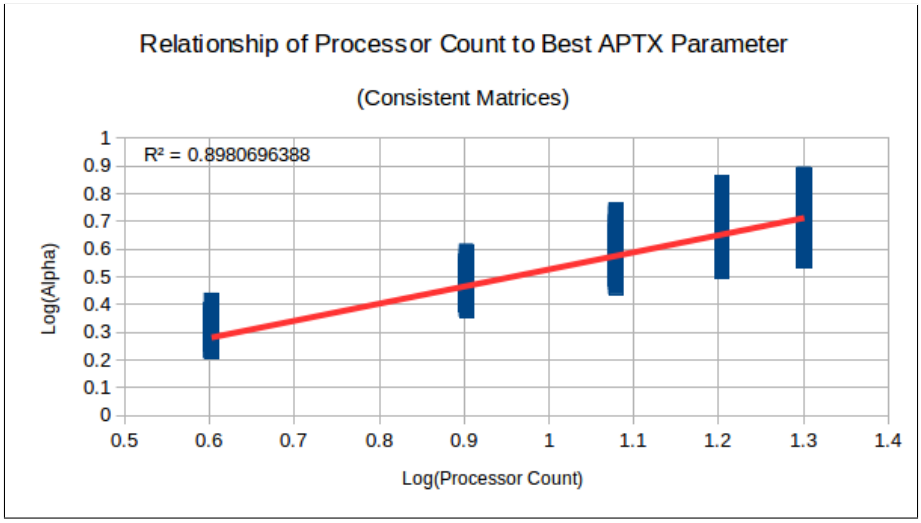

Figure 3.8: Relationship of *procCount* to Best α for APTX, Consistent Systems

From these data, an equation to estimate α was developed. The equation was modeled on the aforementioned predictor-α relationships. This model is shown in (3.7). It consists of the sum of a constant and three powers. Let x_4 represent *minTaskRatio*. Let x_{17} represent *procMeanRatio*. Let x_{19} represent *procCount*).

$$\alpha = b_1 + b_2 x_4^{b_3} + b_4 x_{17}^{b_5} + b_6 x_{19}^{b_7} \tag{3.7}$$

This model and the data were run through the Matlab fitnlm (fit non-linear model) function in order to find good values for $b_i, i \in \{1, 2, ..., 7\}$. The resulting regression function is shown in (3.8). The coefficient of determination is $r^2 = 0.946$, indicating a very good fit between the equation and the observed data.

$$\alpha = -1.9302 + 2.5357 x_4^{-0.0030292} + 0.1609 x_{17}^{0.92276} + 0.27342 x_{19}^{0.77897} \tag{3.8}$$

This function was integrated into the simulator to provide an automatically computed and precise value of α to use with APTX and a consistent ETC matrix. However, further analysis reveals the true predictors of α. For instance, the x_4 term can be reduced to a constant, just as it was for APT and consistent matrices. Performing

this simplification and approximating the other coefficients and powers results in the more simple equation given in (3.9). Again, let x_{17} represent *procMeanRatio* and let x_{19} represent *procCount*).

$$\alpha = 0.61 + 0.16x_{17}^{0.92} + 0.27x_{19}^{0.78} \tag{3.9}$$

Hence, the most meaningful predictor of α for APTX on a consistent system is processor count, as evidenced by the high coefficient. This is as one would expect for a consistent system. Each task "ranks" machines in the exact same way. Therefore, in order to keep load balance, α needs to allow several machines for several tasks. The exact number of these "several" machines is of course a function of the number of machines in the system. Naturally, in order to choose a value of α, the "typical difference" between execution times of a task is also needed. Fundamentally, this is the meaning of task heterogeneity. Hence, the second-best predictor of α is task heterogeneity. In addition, each of these predictors diminishes in its returns as it increases. Adding one more machine to a large consistent system will not offer much of a benefit. Similarly, if task heterogeneity is high, adding just a bit more won't drastically change the ideal threshold.

Note that the constants b_i used for APTX in (3.9) nearly match those used in (3.3) for APT. The independent predictors x_i match exactly. This is expected as the schedulers function in nearly-identical ways. The only difference is that each task is eligible to be mapped to an unlimited number of machines, as opposed to just one or two.

Next, the same 5430 inconsistent experiments from earlier were run again. The same values of α were used, and the same criteria for identifying the best value of α were used, this time with APTX instead of APT. Again, each experiment was simulated with each value $\alpha \in \{1.0, 1.1, 1.2, ..., 8.0\}$. The assumption was again upheld that it would never be a good decision to assign a task to a machine eight times

slower than the fastest one for that task. The same criteria for selecting the "best" value of α for an experiment were used. Again, Octave was used to compute the correlation between each feature and α for potential linear, exponential, logarithmic, and power relationships. Those with $r^2 >= 0.25$ were recorded. Any smaller correlation was not considered to be meaningful. Listing 3.4 shows the text output of the script performing this function. This text was inspected in the same way as for APT experiments.

Listing 3.4: APTX Inconsistent Correlations

```
feat 4,lin: r2=0.284109
feat 4,exp: r2=0.290550
feat 4,log: r2=0.280873
feat 4,pow: r2=0.292474
feat14,lin: r2=0.277161
feat14,exp: r2=0.285678
feat14,log: r2=0.285545
feat14,pow: r2=0.298884
feat17,lin: r2=0.313092
feat17,exp: r2=0.312466
feat17,log: r2=0.315031
feat17,pow: r2=0.315100
```

Only one of each type of measurement was kept. In the case of inconsistent systems running APTX, the measurements were the same as those of inconsistent systems running APT. *taskMeanExtremaRatio* gave the highest correlation between machine heterogeneity and α. This relationship is plotted in Figure 3.9. Similarly, *procMeanRatio* was the most useful measurement of task heterogeneity when in a power relationship. This is plotted in Figure 3.10. Task heterogeneity was determined to be a poor indicator of the best α when running APT on an inconsistent system.

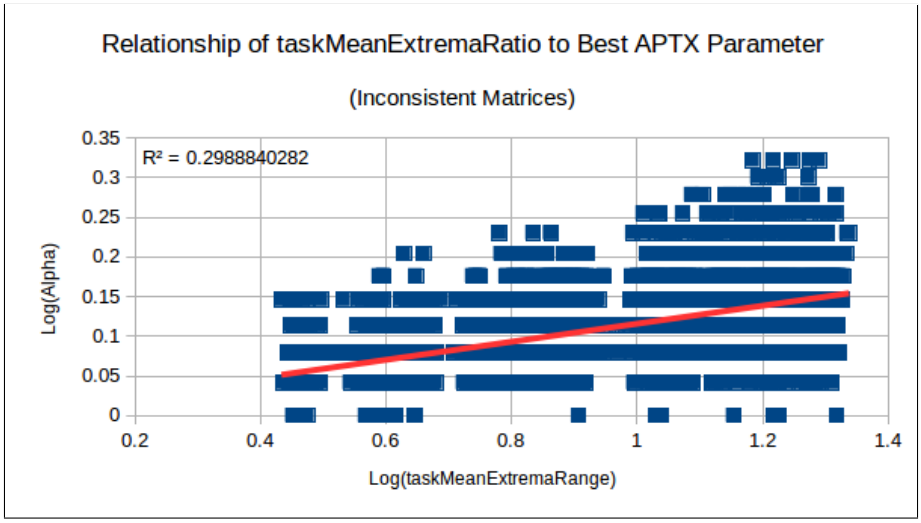

Figure 3.9: Relationship of *taskMeanExtremaRatio* to Best α for APTX

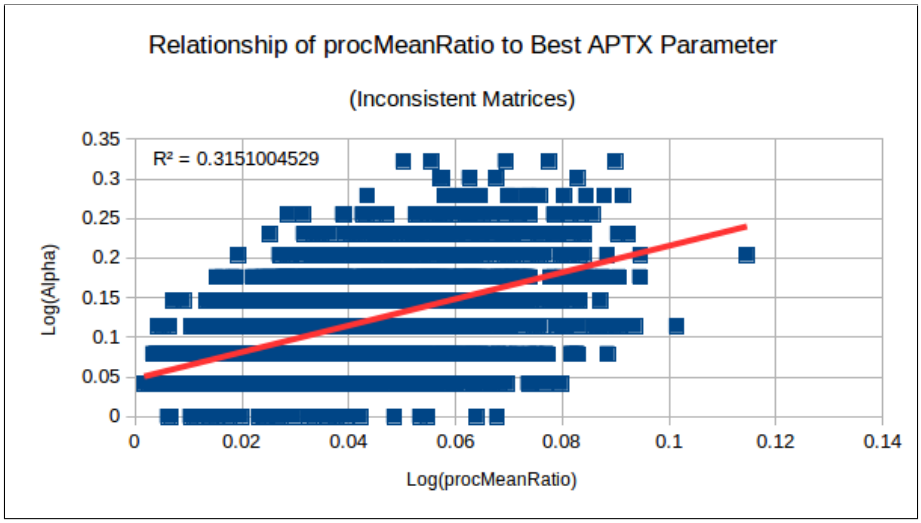

Figure 3.10: Relationship of *procMeanRatio* to Best α for APTX

From these data, an equation to estimate α was developed. The equation was modeled on the aforementioned predictor-α relationships. This model is shown in (3.10). It consists of a constant and and a sum of two powers. Let x_{14} represent *taskMeanExtremaRatio* and let x_{17} represent *procMeanRatio*.

$$\alpha = b_1 + b_2 x_{14}^{b_3} + b_4 x_{17}^{b_5} \tag{3.10}$$

This model and the data were run through the Matlab fitnlm (fit non-linear model)

function in order to find good values for $b_i, i \in \{1, 2, ..., 5\}$. The resulting regression function is shown in (3.11). The coefficient of determination is $r^2 = 0.431$, which is much weaker than that for APTX and consistent matrices, but similar to that for APT and inconsistent matrices.

$$\alpha = -51.652 + 0.060482x_{14}^{0.54288} + 52.609x_{17}^{0.031834} \tag{3.11}$$

This function was integrated into the simulator to provide an automaticallycomputed and precise value of α to use with APTX and an inconsistent ETC matrix. However, further analysis reveals the true predictors of α. For instance, the x_{17} can be approximated to a constant just as it was for APT and inconsistent matrices. As for the x_{14} term, x_{14} does have enough of a range (from about 2.7 to 21.8 in this research) that it should still be considered in spite of its small power. Performing these simplifications results in (3.12).

$$\alpha = 0.92 + 0.06x_{14}^{0.54} \tag{3.12}$$

Hence, the only significant predictor of α for APTX on an inconsistent system is machine heterogeneity. Unlike in a consistent system, the same machines are not the best for every task. Hence, the purpose of α has less to do with setting the number of eligible machines, so the processor count and task heterogeneity become poor predictors.

3.1.3 Search for Best KPB Parameter

The same 5430 consistent experiments from earlier were run this time using KPB as the scheduler. Each $k = \frac{100}{\mu} + 5n$, $n \in \mathbb{N}$, $k \leq 100$ was simulated with each experiment. The value of k which resulted in the earliest finish time was recorded for each experiment. In the case of a tie, the least value of k was used. Next, Octave

was used to compute the correlation between each feature and k for potential linear, exponential, logarithmic, and power relationships. Those with $r^2 >= 0.25$ were recorded. Any smaller correlation was not considered to be meaningful. Listing 3.5 shows the text output of the script performing this function. This text was inspected in the same way as for APT and APTX experiments.

Listing 3.5: KPB Consistent Correlations

```
feat  4,log:  r2=0.280017
feat  4,pow:  r2=0.266484
feat19,lin:   r2=0.360149
feat19,exp:   r2=0.342118
feat19,log:   r2=0.468687
feat19,pow:   r2=0.451783
```

Only one of each type of measurement was kept. For machine heterogeneity, *minTaskRatio* was determined to be the best predictor of k when in a logarithmic relationship. This relationship is plotted in Figure 3.11. Similarly, *procCount* was the best indicator of k when in a power relationship. This is plotted in Figure 3.12. Task heterogeneity was determined to be a poor predictor of k.

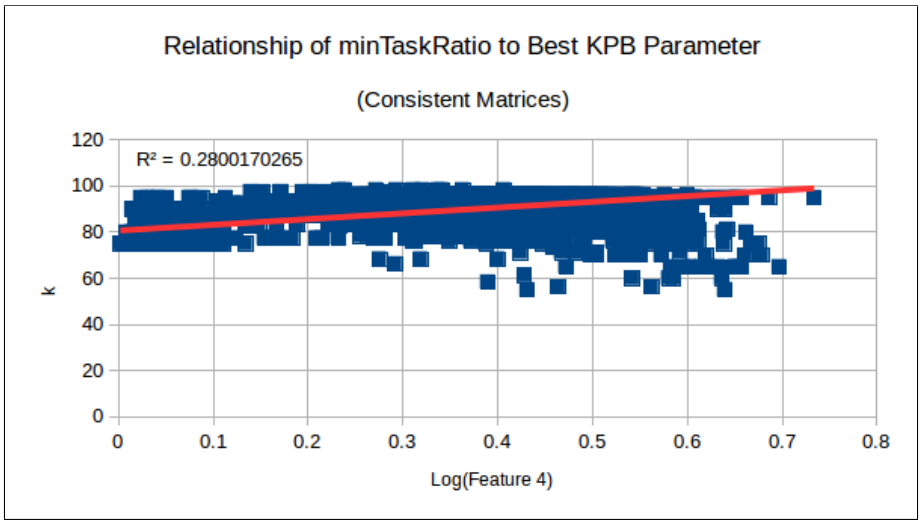

Figure 3.11: Relationship of *minTaskRatio* to Best k for KPB, Consistent Systems

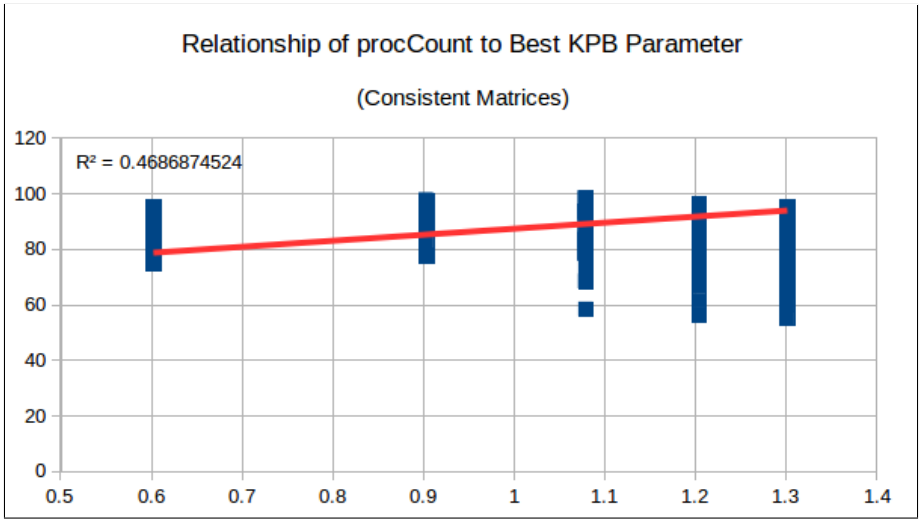

Figure 3.12: Relationship of *procCount* to Best k for KPB, Consistent Systems

From these data, an equation to estimate k was developed. The equation was modeled on the aforementioned predictor-k relationships. This model is shown in (3.13). It consists of a constant, and a sum of two logarithms. Let x_4 be *minTaskRatio* and let x_{19} be *procCount*).

$$k = b_1 + b_2 log(x_4) + b_3 log(x_{19}) \tag{3.13}$$

This model and the data were run through the Matlab fitnlm (fit non-linear model) function in order to find good values for $b_i, i \in \{1, 2, 3\}$. The resulting regression function is shown in (3.14). The coefficient of determination is $r^2 = 0.54$, indicating a somewhat-good fit between the equation and the observed data.

$$k = 54.291 - 14.248 log(x_4) + 18.306 log(x_{19}) \tag{3.14}$$

Each term of this equation is significant, meaning that both machine heterogeneity and machine count significantly impact the ideal k for a consistent system. First, note that the machine count has a negative impact on the calculation of k. This makes sense in KPB, as the number of machines eligible for each task mapping are

a function of machine count μ and k itself. Recall that the size of that subset of machines is $\frac{k \times |M|}{100}$. There is an absolute number of machines that are desirable, so as the number of machines increases, the fraction of machines which are eligible falls relatively. Machine heterogeneity determines what this number of machines should be, as it measures how much worse each machine is than the previous one. Hence, this has a positive impact on the ideal value of k.

Next, the same 5430 inconsistent experiments from earlier were run again. The same values of k were used, and the same criteria for identifying the best value of k were used. Again, Octave was used to compute the correlation between each feature and k for potential linear, exponential, logarithmic, and power relationships. Those with $r^2 >= 0.25$ were recorded. Any smaller correlation was not considered to be meaningful. Listing 3.6 shows the text output of the script performing this function. This text was inspected in the same way as for APT and APTX experiments.

Listing 3.6: KPB Inconsistent Correlations

```
feat 4,lin: r2=0.667669
feat 4,exp: r2=0.720044
feat 4,log: r2=0.765847
feat 4,pow: r2=0.776132
feat 6,lin: r2=0.325544
feat 6,exp: r2=0.383362
feat 6,log: r2=0.337016
feat 6,pow: r2=0.372322
feat14,lin: r2=0.632856
feat14,exp: r2=0.715737
feat14,log: r2=0.739522
feat14,pow: r2=0.763029
feat19,lin: r2=0.780071
feat19,exp: r2=0.799213
feat19,log: r2=0.843128
feat19,pow: r2=0.799745
```

Only one of each type of measurement was kept. For instance, as was the case with APT *minTaskRatio*, *meanTaskRatio*, and *taskMeanExtremaRatio* are all fundamentally measurements of machine heterogeneity. While it is the case that *minTaskRatio* gave the highest correlation of them all in a power relationship, it was actually discarded. Instead, *taskMeanExtremaRatio* was used, as it had nearly as high of a correlation and was used also with APT and APTX. This relationship is plotted in Figure 3.13. *procMeanRatio* was determined to correlate well with k in a power relationship. This is plotted in Figure 3.14.

From these data, an equation to estimate k was developed. The equation was modeled on the aforementioned predictor-k relationships. This model is shown in (3.15). It consists of a constant, a power, and a logarithm. Let x_{14} be *taskMeanExtremaRatio*

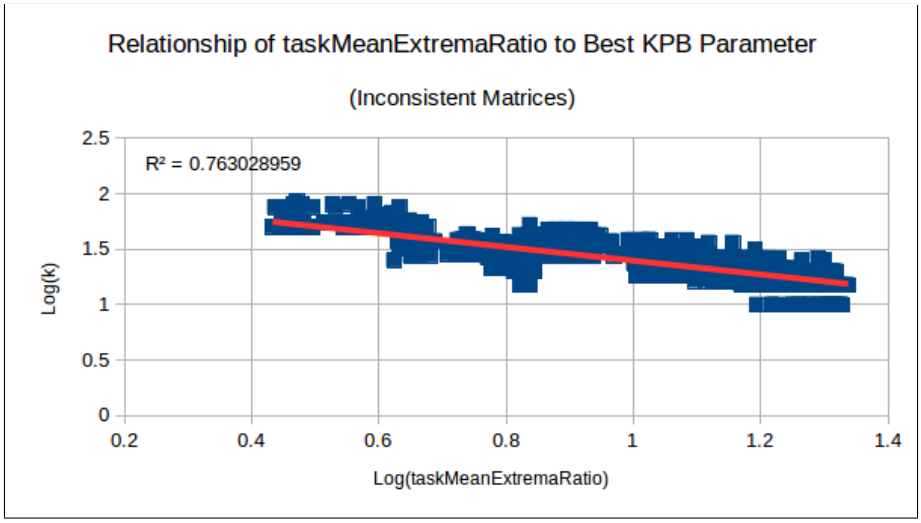

Figure 3.13: Relationship of *taskMeanExtremaRatio* to Best k for KPB

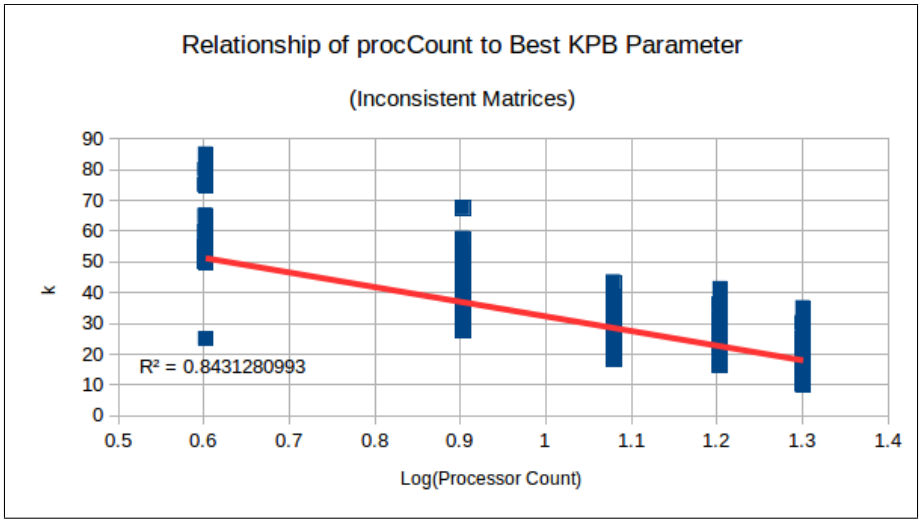

Figure 3.14: Relationship of *procCount* to Best k for KPB

and let x_{19} be *procCount*.

$$k = b_1 + b_2 x_{14}^{b_3} + b_4 log(x_{19}) \tag{3.15}$$

This model and the data were run through the Matlab fitnlm (fit non-linear model) function in order to find good values for $b_i, i \in \{1, 2, 3, 4\}$. The resulting regression function is shown in (3.14). The coefficient of determination is $r^2 = 0.89$, indicating

a very good fit between the equation and the observed data.

$$k = 45.605 - 45.706x_4^{-0.47356} + -13.686log(x_{19}) \tag{3.16}$$

Each term of this equation is significant, meaning that both machine heterogeneity and machine count significantly impact the ideal k for an inconsistent system, just like in a consistent system. The explanation for this is the same. There is an absolute number of machines that are desirable for each task, so as the number of machines increases, the fraction of machines which are eligible falls relatively. Hence, this has a negative impact on the ideal value of k. Machine heterogeneity determines what this number of machines should be, as it measures how much worse each machine is than the previous one. Hence, this has a substantial impact on the ideal value of k.

3.2 Evaluation of Parameter Expressions

Equations (3.2), (3.5), (3.8), (3.11), (3.14), and (3.16) were added to the simulator such that it was then capable of estimating the ideal parameter of each scheduler for each type of matrix. Next, the efficacy of each equation was evaluated. The 5430 consistent experiments and the 5430 inconsistent experiments were run once with each of the six dynamic schedulers — MET, SS, SPN, APT, APTX, and KPB. In the case of the latter three, values of α and k were computed based on the corresponding aforementioned equation. The finish time of each simulation was normalized relative to that of MET (the baseline). Two measures were used to evaluate the efficacy of each scheduler. First is the number of times that scheduler performed the best for an experiment. Second is the average normalized finish time. This says *how much* better (or worse) each scheduler is than MET (i.e., speedup).

The number of times each scheduler performed the best was recorded separately for consistent and inconsistent systems. These results are summarized in Table 3.2.

Table 3.2: Number of Times Each Scheduler Finished Earliest

	Consistent	Inconsistent	Total
MET	0	26	26
SS	1359	714	2073
SPN	1990	808	2798
APT	174	1444	1618
APTX	167	1568	1735
KPB	0	0	0

Bar graphs of the data for consistent and inconsistent systems are shown in Figures 3.15 and 3.16 respectively. Note that while only 3540 experiments were run for each type of system, more than one scheduler tied for some experiments. Hence, column totals may exceed the number of experiments of the corresponding type.

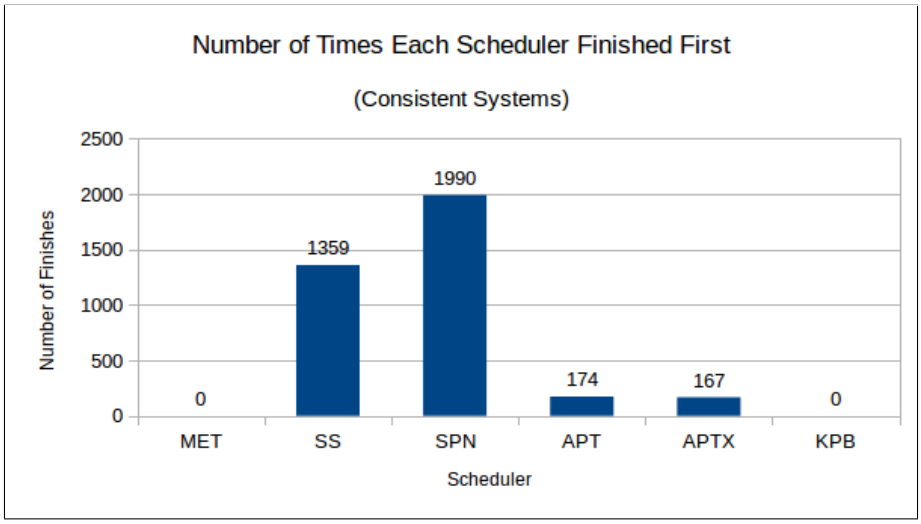

Figure 3.15: Number of Times Each Scheduler Finished Earliest in Consistent Experiments

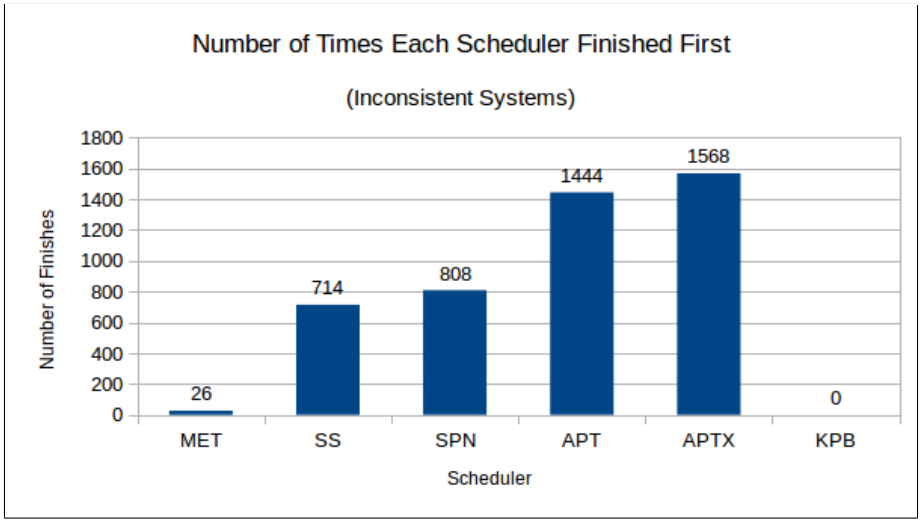

Figure 3.16: Number of Times Each Scheduler Finished Earliest in Inconsistent Experiments

It is clear at this point that KPB and MET are very poor scheduling policies for the types of workloads simulated in this research. Beyond that, the best scheduler depends on whether the system is consistent or inconsistent. For a consistent system, SPN performed the best *most often*, followed by SS. APT and APTX rarely performed the best on consistent systems. However, for inconsistent systems (more realistic in heterogeneous computing), APT and APTX performed better than all other schedulers by a wide margin. Consider the fact that APT and APTX are similar. The two combined cover almost half of the test cases.

The average speedup metric accounts for the result of every experiment and scheduler, and quantifies *how much* better each scheduler is in comparison to MET (and each other). Figure 3.17 shows the speedup of each scheduler relative to MET for consistent systems. Figure 3.18 shows the speedup of each scheduler relative to MET for inconsistent systems.

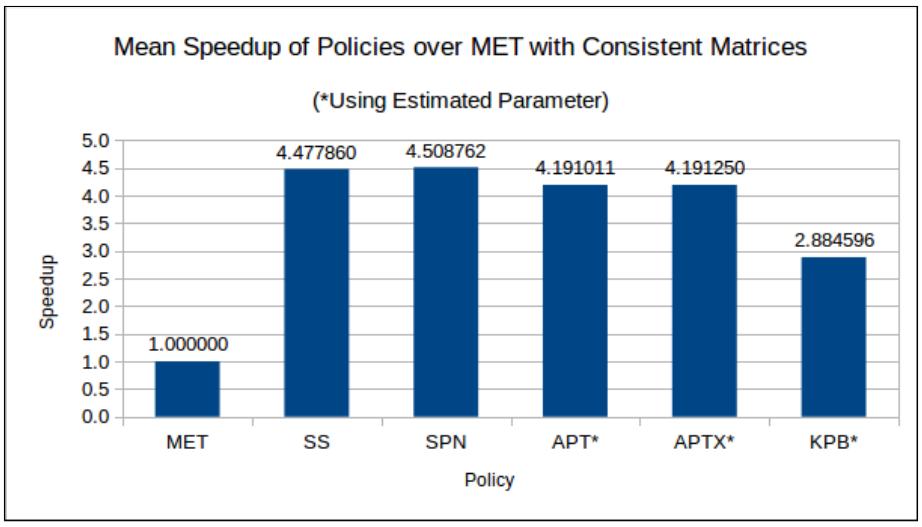

Figure 3.17: Speedup of Schedulers over MET for Consistent Systems

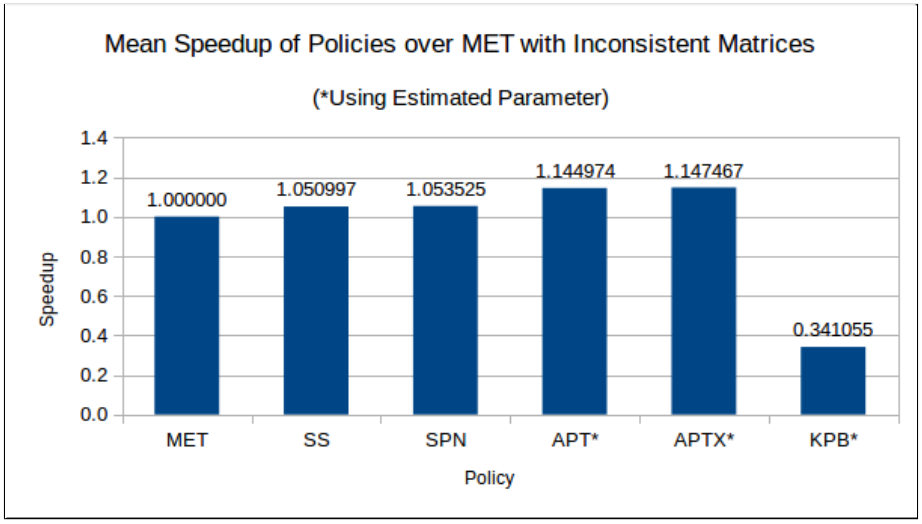

Figure 3.18: Speedup of Schedulers over MET for Inconsistent Systems

As these figures show, SPN provided the greatest average speedup for consistent systems, at 4.51 times faster than MET. This was followed by SS at 4.48. APTX, APT, and KPB (each using the estimated parameters) followed with speedups of 4.19, 4.19, and 2.88 respectively. Overall, MET was by far the worst scheduler for consistent systems, and all of the other schedulers offered drastic improvements.

On the other hand, APTX and APT were the best and second-best schedulers, respectively, for inconsistent systems — the type of configuration that is most common

in heterogeneous computing. APTX had an average speedup of 1.15 over MET. APT had an average speedup of 1.14. This does suggest that APTX may be a slightly better scheduling heuristic, but at the cost of more complexity (i.e., computations) with each mapping considered.

Chapter 4

Conclusions

The key conclusions made from this research are the following:

- For each scheduler, the features of an ETC matrix which best predict an appropriate parameter value were determined.
- SS and SPN were identified as the best schedulers for consistent systems.
- APT and APTX were identified as the the best schedulers for inconsistent systems.
- A methodology for tuning parametric schedulers was demonstrated.

This research explored which characteristics of consistent and inconsistent heterogeneous systems best predict a proper value of α for the APT scheduler, α for the APTX scheduler, and k for the KPB scheduler. Machine heterogeneity correlated well with the ideal parameter of all schedulers on all types of systems, but only on consistent systems and those running KPB did it have a large impact. Task heterogeneity correlated well with α in APT and APTX, but had a strong influence on the ideal value only if the system was consistent. Machine heterogeneity was a strong indicator of the ideal value of k in KPB and α on consistent systems running APT and APTX. A visual summary of these findings is shown in Figure 4.1

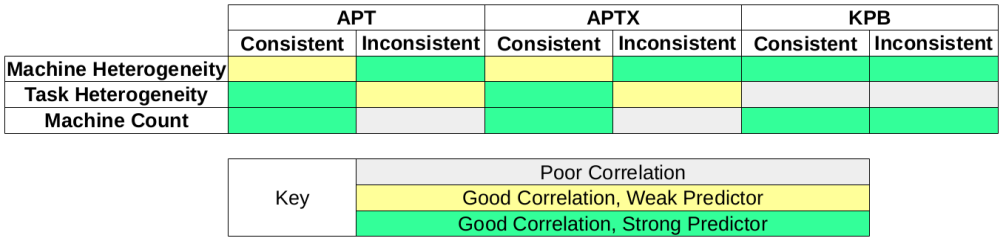

Figure 4.1: Summary of Characteristics that Predicted Best Parameter Values

Furthermore, SS and SPN were identified as good dynamic scheduling heuristics for consistent distributed systems. APT and APTX performed nearly as well on consistent systems and performed the best out of any examined schedulers on inconsistent systems. Given that heterogeneous systems are usually not consistent — due to different hardware architectures matching different types of software parallelism — APT and APTX can reasonably be considered to be the best choices in general. APTX performed slightly better than APT on average, but with more complexity (i.e., overhead). In some applications, this criterion may sway a developer or administrator to one or the other.

While KPB performed better than MET on consistent systems, it was still the second worst for such systems. Furthermore, it was by far the slowest scheduling heuristic to be simulated on inconsistent systems — the more typical type of heterogeneous system. Given these facts, KBP is not recommended.

From a more general standpoint, this research demonstrated a methodology for "tuning" parametric schedulers. It could be performed in the same way for any scheduler that requires a numeric parameter prior to execution. The result of this procedure would be some different equation for that parameter, once again based on machine heterogeneity, task heterogeneity, and machine count. This does not guarantee that the scheduler will be better than existing ones on average, but it does make it more likely that the parameter being chosen for any ETC matrix is appropriate and will lead to best- or near best-case performance.

One key feature of this research is that it applied the simulation model and assumptions from the [12] and [13] scheduler comparison studies to a selection of additional dynamic scheduling heuristics. This places them on a more even ground for comparison. Now APT, APTX, KPB, MET, SS, and SPN can be fairly compared to other scheduling heuristics from the original comparison studies. That being said, there are limitations to the simulation model. The use of uniform random distributions for the generation of ETC matrices may not match the distributions of execution times of actual tasks (kernels) on actual commercially-available machines. If one were to deploy APT (or APTX) to a heterogeneous system in production, one would have to profile the performance of each task on each machine in advance of deployment just to generate the ETC matrices. For best results, one may want to repeat the procedure of this research in order to develop a new equation for α, better suited for the tasks and machines being used.

4.1 Future Work

There are many opportunities for future research based on the findings of this thesis. Three such possibilities are identified here.

- Simulation of realistic workloads based on mappings of actual tasks to actual processors
- Hardware implementation of APT or APTX
- Cybersecurity: Identifying anomalies by monitoring α

This research demonstrated the viability of parameterized schedulers — especially APT and APTX — for scheduling tasks on heterogeneous systems in a simulation model. This opens up a number of interesting research opportunities. One variation of this work would include the simulation of realistic workloads. This would involve

profiling the performance of various tasks on various machines (as was done in [19]) and updating ETC matrices to reflect them.

Another possibility would be to proceed to a hardware implementation of one of these schedulers — possibly APT or APTX. A natural first attempt for any sort of hardware implementation of a logical design is on an FPGA. Furthermore, hardware industry leaders have introduced SoCs with different CPUs, GPUs, and FPGA fabric, such as Xilinx's Zynq [28, 29]. It would make sense to implement and test the scheduler on the FPGA fabric of such a device.

This research also presents opportunities for cyber-security research. In particular, scheduler behavior could be used to detect anomalous behavior (i.e., malware) on a heterogeneous system. For instance, suppose the ETC matrix is updated dynamically in accordance with the current (backlog) of a workload. The contents of the ETC matrix inform the value of α being used. If α were to stray outside of some expected range, that could indicate an anomalous workload (i.e., unexpected tasks are present).

Bibliography

[1] S. Mittal, "Power management techniques for data centers: A survey," 04 2014.

[2] A. Khokhar, V. Prasanna, M. E. Shaaban, and C.-L. Wang, "Heterogeneous computing: Challenges and opportunities," vol. 26, pp. 18 – 27, 07 1993.

[3] M. Maheswaran, T. Braun, and H. Siegel, "Heterogeneous Distributed Computing," *Encyclopedia of Electrical and Electronics Engineering*, vol. 8, pp. 679–690, 1999.

[4] R. Freund and H. Siegel, "Heterogeneous Processing," *IEEE Computer*, vol. 26, no. 6, pp. 13–17, 1993.

[5] Z. Cui, Y. Liang, K. Rupnow, and D. Chen, "An accurate gpu performance model for effective control flow divergence optimization," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, May 2012, pp. 83–94.

[6] T. Braun, H. Siegel, N. Beck, L. Blni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, and B. Yao, "A Taxonomy for Describing Matching and Scheduling Heuristics for Mixed-Machine Heterogeneous Computing Systems," *IEEE Workshop on Advances in Parallel and Distributed Systems*, pp. 330–335, 1998.

[7] D. Fernndez-Baca, "Allocating Modules to Processors in a Distributed System," *IEEE Transactions on Software Engineering*, vol. 15, no. 11, 1989.

[8] U. M. Mirza, M. Arslan, G. Cedersj, S. Sulaman, and J. Janneck, "Mapping and scheduling of dataflow graphs - a systematic map," vol. 2015, 11 2014.

[9] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Transactions on Computers*, vol. C-36, no. 1, pp. 24–35, Jan 1987.

[10] S. Karia, "Alternative Processor within Threshold: Flexible Scheduling on Heterogeneous Systems," Master's thesis, Rochester Institute of Technology, March 2017.

[11] M. Maheswaran, T. D. Braun, and H. J. Siegel, "High-performance mixed-machine heterogeneous computing," in *Parallel and Distributed Processing, 1998. PDP '98. Proceedings of the Sixth Euromicro Workshop on*, Jan 1998, pp. 3–9.

[12] T. Braun, H. Siegel, N. Beck, L. Blni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, D. Hensgen, and R. Freund, "A Comparison Study of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems," 2000.

[13] M. Maheswaran, A. S., H. Siegel, , D. Hensgen, and R. Freund, "Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems," in *Heterogeneous Computing Workshop, 1999. (HCW '99) Proceedings. Eighth*, 1999, pp. 30–34.

[14] T. D. Braun, H. J. Siegel, N. Beck, L. L. Blni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, "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, pp. 810 – 837, 2001.

[15] C. W. Fletcher, I. A. Lebedev, N. B. Asadi, D. R. Burke, and J. Wawrzynek, "Bridging the gpgpu-fpga efficiency gap," in *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2011, pp. 119–122.

[16] H. M. Hussain, K. Benkrid, H. Seker, and A. T. Erdogan, "Fpga implementation of k-means algorithm for bioinformatics application: An accelerated approach to clustering microarray data," in *2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, June, pp. 248–255.

[17] D. Chen and D. Singh, "Invited paper: Using opencl to evaluate the efficiency of cpus, gpus and fpgas for information filtering," in *22nd International Conference on Field Programmable Logic and Applications (FPL)*, Aug 2012, pp. 5–12.

[18] A. P. D. Binotto, D. Doering, T. Stetzelberger, P. McVittie, S. Zimmermann, and C. E. Pereira, "A cpu, gpu, fpga system for x-ray image processing using high-speed scientific cameras," pp. 113–119, 2013.

[19] S. Skalicky, S. Lopez, and M. Lukowiak, "Distributed execution of transmural electrophysiological imaging with cpu, gpu, and fpga," in *2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, Dec 2013, pp. 1–7.

[20] R. Armstrong, D. Hensgen, and T. Kidd, "The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions," in *Heterogeneous Computing Workshop, 1998. (HCW 98) Proceedings. 1998 Seventh*, Mar 1998, pp. 79–87.

[21] M. M. Eshaghian and Y. C. Wu, "Mapping heterogeneous task graphs onto heterogeneous system graphs," in *Heterogeneous Computing Workshop, 1997. (HCW '97) Proceedings., Sixth*, Apr 1997, pp. 147–160.

[22] R. F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. D. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel, "Scheduling resources in multi-user, heterogeneous, computing environments with smartnet," in *Heterogeneous Computing Workshop, 1998. (HCW 98) Proceedings. 1998 Seventh*, Mar 1998, pp. 184–199.

[23] C.-K. Luk, S. Hong, and H. Kim, "Qilin: Exploiting parallelism on heterogeneous multiprocessors with adaptive mapping," in *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 42. ACM, 2009, pp. 45–55.

[24] H. Arabnejad and J. G. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 3, pp. 682–694, March 2014.

[25] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, Mar 2002.

[26] S. Skalicky, S. Lopez, M. Lukowiak, and C. A. Wood, "Graph-based design methodology for pipelined architectures," July 2016.

[27] C. Liu and S. Yang, "A heuristic serial schedule algorithm for unrelated parallel machine scheduling with precedence constraints," vol. 6, pp. 1146–1153, 06 2011.

[28] [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc.html

[29] [Online]. Available: https://www.altera.com/products/soc/overview.html