Robust Resource Allocation of Independent Tasks in Heterogeneous Computing Systems via Probabilistic Task Pruning

A Thesis

Presented to the

Graduate Faculty of the

University of Louisiana at Lafayette

In Partial Fulfillment of the

Requirements for the Degree

Master of Science

James A. S. Gentry

Summer 2018

Robust Resource Allocation of Independent Tasks in Heterogeneous Computing Systems via Probabilistic Task Pruning

James A. S. Gentry

APPROVED:

Mohsen Amini Salehi, Chair
Assistant Professor of Computer Science
The Center for Advanced Computer Studies

Nian-Feng Tzeng
Professor of Computer Science
The Center for Advanced Computer Studies

Miao Jin
Associate Professor of Computer Science
The Center for Advanced Computer Studies

Sheng Chen
Assistant Professor of Computer Science
The Center for Advanced Computer Studies

Mary Farmer-Kaiser
Dean of the Graduate School

To my parents; to my wife Erin.

Acknowledgments

I sincerely thank my supervisor, Professor Mohsen Amini Salehi, for his constant encouragement and passion for computer science and, especially, for his guidance and support. Thanks to my thesis committee, Miao Jin, Nian Feng Tzeng, and Sheng Chen. Finally, thanks goes to the Center for Advanced Computer Studies and the Graduate School at the University of Louisiana at Lafayette for their support and guidance.

Table of Contents

Chapter 1: Introduction

Distributed Heterogeneous Computing (HC) systems can be described based on qualitative and quantitative differences between their machines [1]. Qualitative machine heterogeneity describes architectural differences between machines (*e.g.,* CPU versus GPU versus FPGA [2, 3, 4]) whereas quantitative machine heterogeneity describes the performance differences within a given architecture (*e.g.,* processors with different clock speeds). For instance, Amazon Cloud (AWS) offers CPU-Optimized, Memory-Optimized, and Accelerated Computing (GPU and FPGA) Virtual Machine (VM) types and within each type, they offer various performances with different prices [5]. In the literature, these are also known as *inconsistent machine heterogeneity* and *consistent machine heterogeneity* [1, 6, 7, 8], respectively.

In addition to machine heterogeneity, diversity can be present in the tasks arriving to an HC system. As with machine heterogeneity, task heterogeneity can be categorized as inconsistent versus consistent heterogeneity. In many current HC systems (*e.g.,* [9, 10, 11]), both types of machine and task inconsistent heterogeneities exist at the same time [12]. In such a system, each task type can have different execution times on different machines type of the system. For instance, machine A may be faster than machine B for task 1 but slower than other machines for task 2. A more specific example can be a task with graphical demands that runs faster on (*i.e.,* matches better with) a GPU whereas another task with numerous memory accesses runs faster on a CPU.

This mixture of heterogeneities creates uncertainty in execution time of each

task type on each machine type [13]. The uncertainty, if not captured, hinders predictability, and subsequently, efficiency of task scheduling [14]. *Robustness* is defined as the degree to which a system can maintain a given level of performance in the face of uncertainty [14]. The overall *goal* of this study is to maximize the robustness of an HC system.

1.1 Motivations

The motivation for this research comes from an inconsistently HC system used for processing live video streaming services [15] (*e.g.,* Google's YouTube Live and Twitch.tv). In these services, video content is initially produced in a source format by the content creator (camera). To support diverse viewers' display devices, the video content has to be processed (*i.e.,* transcoded) to adapt the characteristics of the viewers' display devices [16].

In an HC system processing live video streams, the task for converting compression standard (codec) is categorically different than the task for converting the video resolution (*i.e.,* inconsistent task heterogeneity) [8]. Within each category of tasks (termed *task type*), tasks with different data sizes can be considered as consistent task heterogeneity. For instance, tasks for converting the codec of different videos are considered consistently heterogeneous. It has been shown that an HC system should be deployed to efficiently carry out different types of video processing (*e.g.,* changing resolution, changing encoding format) and ensure a real-time flow of processed video streams to the viewers [8]. In such an HC system, there exists inconsistent task heterogeneity in form of different task-types; and inconsistent machine heterogeneity in

form of different machine-types (*e.g.,* CPU-based and GPU-based).

The intensity of tasks arriving to the HC system (known as oversubscription level) also varies. Managing this oversubscription is a key to ensure the number of tasks that meet their deadline is maximized (*i.e.,* the system remains robust). As there is no value in executing tasks that have missed their individual deadlines, they are dropped from the HC system.

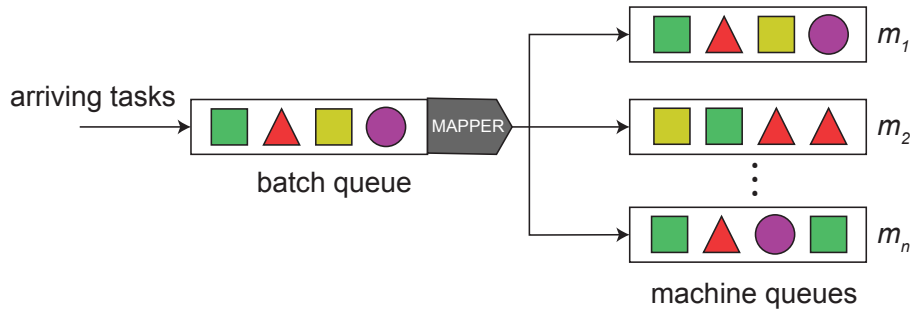
In addition to live video streaming, another motivation can be monitoring and mitigating an oil spill via edge computing in a smart oilfield: Streams of image data from cameras in UAVs [17], and from embedded in the platform are monitored for evidence of anomalies indicating the direction of movement of an oil spill. Sensors in the drill pipe offer real-time data to be processed to better understand the movement of fluids underground and on the surface [18]. Both of these tasks require processing of streams of data with hard and quick deadlines [19] (*e.g.,* hourly or daily). The tasks are structurally different (*i.e.,* heterogeneous), and benefit from being processed on different types of (*i.e.,* heterogeneous) machines. Whereas the processing of large volumes of images is best handled by parallel GPU-based machines [20], the models that predict flows of oil benefit from FPGA [21]. Putting all of the above together can be demanding in terms of memory [22]. Essentially this thesis is concerned with any system that requires liveness in its data, but is also tolerant of portions of the data being dropped.

1.2 Research Problem and Objectives

In this study, as shown in Figure 1.1, arriving tasks are placed in a batch queue before being mapped to available machines. Each arriving task has a hard individual

deadline. That is, there is no value in executing a task that has missed its deadline and it must be dropped (*i.e.,* discarded) by the system [23, 24, 25]. In the circumstance that the HC system is heavily loaded, it is impossible for all tasks to complete before their deadlines, *i.e.,* the system is *oversubscribed*. The robustness of the HC system is measured based on the number of tasks meeting their deadlines and consider that as the performance metric. Therefore, the more specific goal of this work is: maximizing the number of tasks meeting their deadlines (hereafter, termed *task success*). This goal must be met in the presence of uncertainty in task execution time and oversubscription.

Figure 1.1. Arriving tasks are queued. Batches of heterogeneous tasks are mapped to inconsistent heterogeneous machines.



To meet this goal, the scheduler (aka mapper) of the HC system, which maps tasks to machines, must be aware of machine and task heterogeneity [26], and must harness this awareness to overcome with the unpredictability of the system. In light of the hard deadline constraint, when a task fails to succeed, the time spent executing it is wasted. Importantly, the wasted processing time causes delay in execution of tasks waiting in the queue, increasing the likelihood of them missing their deadlines, hence, decreasing the system robustness. To prevent this, the system should avoid executing tasks are not likely to succeed. The question is: *what is the probability that makes a*

task worth executing?

1.3 Methodology Overview

To address this problem, in this research, a suitable probability to map tasks to machines is determined, so that the computational time is not wasted to process tasks with low chance of success. The scheduler needs to calculate the probability of meeting deadline (success) for a given task, and depending on the probability, decides whether or not to map the task. While the tasks whose probability of success is greater than the suitable probability are mapped to machines, the ones whose chance of success is less than the suitable probability can be pruned in two ways: (A) they can be *deferred* from the mapping decision (*i.e.,* their mapping is deferred to the next mapping event); (B) they can be *dropped* from the system. Deferring mapping of the task is done with the hope that in the next mapping event the task gets a higher probability of success (*e.g.,* because of mapping to a better matching machine). Dropping, however, is performed to alleviate the oversubscription, when the HC system is heavily loaded. The rationale for dropping is that, in an oversubscribed system, sacrificing (dropping) tasks with low chance of success can increase the likelihood of other waiting tasks succeed, hence, improving the robustness. Nonetheless, the question arises: in what level of oversubscription the mapper should become more aggressive, transitioning from only task deferring to both task deferring and dropping?

Mapping tasks in an HC system has been shown to be an NP complete problem [27, 28], hence, numerous heuristics have been developed (*e.g.,* [29, 7, 14, 30, 31, 13, 32, 33]) to maximize robustness of the HC systems. However,

none of these heuristics take task deferring and dropping into consideration. Therefore, in this research, a new heuristic is proposed that considers task success probabilities, deferring, and dropping.

To understand the efficacy of the probabilistic deferring and dropping hypothesis, in this research, a simulation study is conducted. The robustness of the proposed pruning-aware mapping method is compared against other existing heuristics, under varying workload conditions. Simulation results approves the hypothesis and shows that the proposed method increases robustness of the HC system when compared with other mapping heuristics used in the literature. This happens, particularly, when the level of oversubscription in the HC system increases.

1.4 Thesis Organization

This thesis is organized into chapters as follows:

- Chapter 2 is a survey of related works in the literature. Research on dynamic scheduling, modeling uncertainty, heterogeneity, stochastic robustness, the dropping of tasks, is presented. The major contributions of each will be presented, as will the relationship to this work, positioning this thesis amongst the presented research.

- Chapter 3 presents the mathematical model of task-execution uncertainty used in this work: Calculating robustness of task-mapping, the convolution of discrete probability mass functions, and convolution in the presence of task-dropping are detailed. explains the system model used in this work. Each component is

detailed, and the connections are explained.

- In Chapter 4, a probabilistic task pruning mechanism is developed that is responsible for deferring the mapping of tasks, as well as the dropping of queued and executing tasks that are unlikely to succeed. A scheduling heuristic is designed that harnesses the pruning mechanism. The results of a series of experiments are examined.

- Chapter 5 examines the effects of applying the pruning mechanism (in whole and in part) developed in Chapter 4 to a series of preexisting batch and immediate mode heuristics.

- Chapter 6 refines the pruning mechanism from Chapter 4 by decoupling the deferring portion of the pruning mechanism from the dropping portion, as well as examines the effects of other changes such as a dynamic, per-task, pruning threshold.

- Chapter 7 concludes the work, discusses results, and notes directions of further research.

Chapter 2: Survey of Related Literature

2.1 Overview

This chapter provides a survey of other research works undertaken in the fields most related to this work and position the contribution of our works against them. This thesis builds upon works in the field of task scheduling, heterogeneous computing, stochastic robustness, dynamic resource allocation, task dropping, and task deferring. Each of these will be discussed and positioned against below.

2.2 Task Scheduling

At it's heart, in distributed computing, task scheduling involves mapping tasks to available machines. Tasks can depend on each other or may be completely independent in the order of their execution. They sometimes have deadlines, and sometimes do not. Tasks can be scheduled either off-line or on-line (*i.e.,* statically or dynamically). In this thesis, the tasks are independent, have hard deadlines, and arrive and are scheduled dynamically.

Paragon [34] is an immediate mode scheduling system for large-scale heterogeneous data centers. It uses singular value decomposition to classify incoming tasks on their heterogeneity, as well as their interference level for co-scheduled tasks. This is accomplished via the matching of information from historical data, with small sample runs of the task. These classifications are used in a greedy algorithm to select a list of candidate resources first based on interference, and from that the best fit based on heterogeneity [35]. This work is different than ours in that the mapping heuristics are immediate mode using scalar, as opposed to probabilistic, execution times to make

decisions. The performance metrics are also different, as Paragon is concerned with speedup, as there are no deadlines to miss.

To reduce energy consumption of HC systems, an stochastic task mapping method for uncertain task execution time is proposed in [30]. Bag of Tasks (BoT) jobs with a deadline constraint for the whole job are considered. They apply a linear programming technique to balance the makespan time and the energy consumption of the BOT job. The proposed mapping method improves the probability that both the deadline and the energy consumption constraints are fulfilled. Our research is different in several aspects. Most importantly, we consider individual deadlines and our aim is to rise the probability of meeting those individual deadlines. In addition, we focus on the impact discarding tasks in the system.

Tumanov *et al.,* investigate a scheduler that uses available task execution time information and task affinity to make global decisions about scheduling tasks in an HC system using mixed integer linear programming [36]. The scheduler, Tetrisched, fits into a YARN and MapReduce framework to globally make use of this information, both to plan ahead, and to make adaptations to its plans as tasks are entered into the system. In contrast, our system uses a similar set of information to make greedy choices about mapping short-running tasks to machines in batches. Whereas we defer tasks in an effort to match affinity, and consider dropping tasks to alleviate oversubscription, Tetrisched attempts to mitigate the uncertainty of the types of incoming tasks by adapting the tentative scheduling plan, on each new scheduling cycle, in an effort to match jobs with high affinity resources. The focus is on long-running jobs with strict

deadlines, and consider both the total number of tasks meeting their deadlines, as well as the number of reserved and best-effort tasks meeting their deadlines, whereas in our study, we consider smaller liveness-oriented tasks with no reservation but with strict individual deadlines.

2.3 Heterogeneous Computing

Broadly speaking, distributed computing can be divided into to main categories: homogeneous and heterogeneous. Homogeneous computing involves a number of identical resources tied together, such that a given task will have the same performance characteristic on each. Heterogeneous computing on the other hand involves a number of different resources tied together, such that a given task may perform better or worse, depending on the machine it is running on.

Khemka *et al.,* investigate resource management in oversubscribed heterogeneous systems in [23]. A parameterized method of utility function creation from priority, utility class, and urgency is tested. An ETC matrix with deterministic execution times is used, whereas we model the times probabilistically. Preemption of tasks that have been assigned to a machine is not considered. Similar to our work, the batch mode heuristics in this work places tasks into a virtual queue and dropping tasks from the system is practiced. However, unlike our approach of probabilistically determining if a task should be dropped, dropping occurs after a task has reached some threshold of utility.

In another work [8], Li *et al.,* provide a model to configure and provision an inconsistently HC system to process an inconsistently heterogeneous set of video

streaming tasks. Because they consider Video On Demand, they do not consider task dropping, however, our motivation in this study is live video streaming tasks that have hard deadlines that implies task dropping.

2.4 Stochastic Robustness

Robustness is the health of a system. This thesis uses a stochastic robustness measure to make decisions based on a predictive model of the health of its system based on the historical data on hand.

In [13], Shestak *et al.,* lay the groundwork for the use of probability density (PDF) and mass functions (PMF) to model the execution time of tasks. This is in contrast to the prevalent use of scalar, deterministic estimates of task execution times. The work in this study shows the use of such in the case of static resource allocation techniques. The method for convolution of these execution times to form completion times for a queue of tasks is established. This thesis builds upon the static use of PMFs and robustness measure, in a dynamic fashion, while adding the conditions of tasks being dropped probabilistically from executing and pending tasks.

2.5 Dynamic Resource Allocation

This thesis is concerned with online resource allocation for independent tasks. The problem of dynamic resource management was examined in [37]. Machovec *et al.,* create scheduling heuristics designed to maximize the utility earned by an HC. The monotonically declining utility function for each task is a function of the tasks completion time, and acts as deadline. When tasks have reached zero possible utility, they are dropped from the system. The authors consider a system where critical tasks

can preempt executing tasks. The preempted task's is resumed after the critical task completes its execution. This differs from our work in that we consider a system wherein the data constraints are such that the overhead of preemption of tasks is too costly. Machovec *et al.,* use ETC matrix to model execution times, but assume deterministic, as opposed to stochastic, execution times. This is different than our Bayesian probability approach on the handling the compound uncertainty of queued tasks. Another difference in our work is our assumption of hard deadlines and thereby maximizing the number of tasks completed ontime, as opposed to maximizing monotonically declining utility functions.

In [38], Malawski *et al.,* evaluate dynamic scheduling of deadline-and-cost constrained tasks in IaaS clouds. The work is focused on provisioning homogeneous VMs, whereas we focus on static heterogeneous machines. The goal of the study is maximizing the high priority workflows completed, within a deadline, while remaining under budget. Algorithms are introduced that are aware of the workflow, leading to higher performance, and they drop workflows that would result in a loss of high priority tasks completing. We use a similar tactic of dropping tasks to maximize the robustness of our system, however our metrics for success are different, as we consider tasks with individual deadlines, and all with the same priority.

Scalar estimates of runtime are used, however inaccurate estimation is examined, as well as uncertain VM provisioning time. The study also investigates static scheduling, and while it performs very well under ideal conditions, as uncertainty in execution and VM provisioning times are introduced, the static scheduling is unable to

contend with the uncertainty as well as the dynamic algorithms do. Though in a different domain, this is akin to our attempt to mitigate what turn out to be poor batch scheduling decisions by dropping tasks that fail to meet our probabilistic robustness thresholds.

2.6 Task Dropping

In certain scenarios, it is best to stop work on some tasks to use resources to complete other tasks. When tasks with deadlines overrun their deadlines, they can be dropped. This is shown in [14], which studies maximizing the robustness of dynamic resource allocation for tasks with stochastic execution times. Salehi *et al.,* model the stochastic nature of the heterogeneous task-types on heterogeneous machine-types using a matrix of probability mass functions (PMFs). A mathematical model for calculating the completion time of stochastically modeled tasks in the presence of task dropping is provided. This thesis builds upon the matrix of PMFs, and the completion time calculations to enable probabilistic dropping, as in this paper, the tasks are only dropped after their deadlines have passed.

In [15] Li *et al.,* propose a task-scheduling method for a cloud-based live video stream processing service. Their system is responsible for transcoding the video, in chunks, to meet user demand and when in an oversubscribed state, they drop transcoding tasks that have passed their deadline. The work touches on dropping past-deadline tasks in its scheduling section, there is no attempt to predict which tasks will fail to meet their deadlines, or to drop those tasks. While this thesis uses some of the problems faced when live-streaming as a motivation, the robustness-boosting ideas

and methods herein can be applied to any system with droppable tasks that have hard deadlines.

2.7 Task Deferring

Sometimes, the best course of action is no action. In [39], Li *et al.,* investigate static list-scheduling strategies that take deferring tasks until a higher affinity machine opens up into consideration. The authors of [40] explore the tactic of delaying the scheduling of tasks to achieve better locality in clusters, minimizing delay.

In [41] Malone *et al.,* implement a distributed scheduling system where processes bid on resources in a market: 'Enterprise'. Among their results, they find that 'lazy assignment', or deferring task scheduling as long as possible, yields the best results. The performance versus 'eager assignment' increases as error in task-length estimation increases, and the performance increase is larger in heterogeneous environments than in homogeneous environments. This increase in performance from 'lazy assignment', where the task is assigned to a resource only when that resource is capable of immediate computation, forms the foundation for the idea of deferring tasks based on a necessary probability of successful completion.

2.8 Summary

Though task scheduling has been extensively studied in the past, both in terms of homogeneous and heterogeneous computing scenarios, less attention has been given to studying the effects of dropping and deferring tasks, and as far as can be determined, probabilistically dropping and deferring tasks in an effort to increase system robustness has not been studied, therefore this thesis is dedicated to investigating the impact of

probabilistic pruning (task dropping and deferring), particularly, in heterogeneous environments. Before detailing the probabilistic pruning mechanism in Chapter 4, the next chapter explains the theoretical backgroundneeded for the rest of this thesis.

Chapter 3: Background

3.1 Bayesian Statistics for Resource Allocation

In the literature, the expected execution times of different task-types on different machine-types are maintained in an Expected Time to Compute (ETC) matrix [6]. These ETC matrices generally contain the mean execution time, and do not account for consistent task heterogeneity (*e.g.,* those arising from data-size differences across different tasks, environmental factors such as neighboring loads, task switching overhead, etc). The stochastic nature of the execution time of tasks that arises from these factors is modeled via Probability Mass Functions (PMF) called a *Probabilistic Execution Time* (PET) [14]. In an inconsistently HC system, a PET matrix is maintained to describe probabilistic execution time of each task-type on each machine-type. The PET matrix is used by the task mapper to optimally map the tasks to machine. In practice, these PMFs can be obtained from historic execution time information of task-types on each machine in an HC system and modeling them via a histogram [42]. The PET matrix is assumed to be available in our HC system.

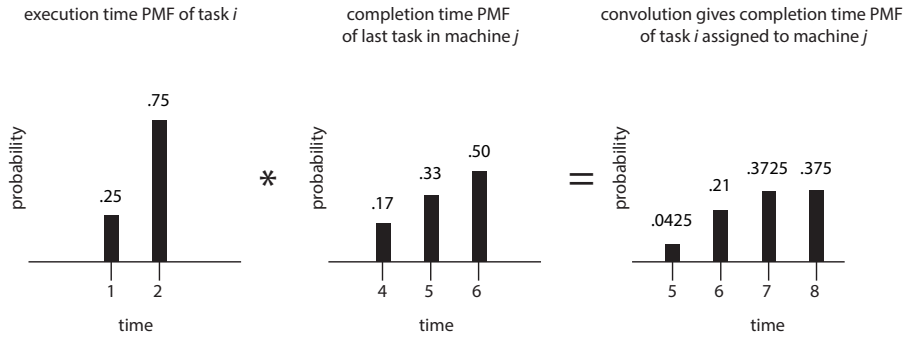
Each entry (i, j) of PET is a PMF represents the "execution time" of task-type i on a machine-type j. When a task i is given a start time on idle machine j, the execution time PMF in $PET(i, j)$ becomes a "completion time" PMF of task i on machine j, denoted *Probabilistic Completion Time* $PCT(i, j)$ [14].

In case machine j is not idle (*i.e.,* it has tasks already executing or pending) and task i arrives, the PCT of the last task in machine j and $PET(i, j)$ are convolved to form $PCT(i, j)$ [43]. This new PMF accounts for all of the execution times of the tasks

ahead of task i in the machine j queue. For example, in Figure 3.1 a task is added to a machine, and its PET is convolved with the PCT of the last task on that machine to form the PCT of the arrived tasks on that machine.

In order to calculate the completion time distribution (CTD) to evaluate possible task-machine mapping decisions, the following is a formalization of the probabilistic task completion time convolution model from [14]. For any CTD generation, the presence of task dropping needs to be taken into account, and if present, the type of task dropping must also be accounted for.

Figure 3.1. Probabilistic Execution Time (PET) of an arriving task is convolved with the Probabilistic Completion Time (PCT) of the last task on a machine to form the PCT for the arriving task on the assigned machine



As shown in Equation 3.1, when task dropping is not permitted, *i.e.,* when all mapped tasks must execute to completion, the formula for calculating the CTD of the entire queue is a simple convolution of the two distributions.

$$C_i(x) = \sum_{k=1}^{k<x} [p_i(k) * C_{i-1}(x-k)] \tag{3.1}$$

When dropping of pending tasks is allowed, the formulation becomes slightly more involved. Equation 3.2 is used by Equation 3.3. Each impulse that is past the deadline for each task becomes zero.

$$f(x,k) = \begin{cases} 0, & \forall(x-k) \geq \delta \\ p_i(k) \times C_{i-1}(x-k), & \forall(x-k) < \delta \end{cases} \tag{3.2}$$

$$C_i^{pend}(x) = \begin{cases} \sum_{k=1}^{k<x} f(x,k) + C_{i-1}(x), & \forall x \geq \delta \\ \sum_{k=1}^{k<x} f(x,k), & \forall x < \delta \end{cases} \tag{3.3}$$

Equation 3.4 is similar to the previous, but allows for the impulses of tasks who are currently executing to be zero as well, when the deadline passes.

$$C_i^{evict}(x) = \begin{cases} \sum_{y=x}^{y<\infty} C_i^{pend}(y) + C_{i-1}(x), & \forall x = \delta \\ C_i^{pend}(x), & \forall x > \delta \\ \sum_{k=1}^{k<x} f(x,k), & \forall x < \delta \end{cases} \tag{3.4}$$

Equation 3.5 shows how the robustness for a given task on a given machine is calculated.

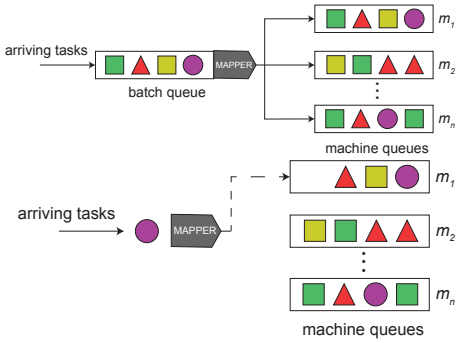
$$p_i(\delta_i) = \sum_{k=1}^{k<\delta_i} C_i(x) \tag{3.5}$$

3.2 System Model Overview

In the system, heterogeneous tasks arrive with individual deadlines, and are entered into a batch queue of unmapped tasks. A *task's robustness* is defined as its probability of completing before its deadline. That is, for task i with deadline δ_i and machine j, $robustness(i, j) = \mathbb{P}(PCT(i, j) \leq \delta_i)$.

A mapping event occurs upon arrival of a new task to the system or when a task gets completed. Before the mapping event, tasks which have missed their deadlines are dropped (*i.e.,* removed) from the system. The mapper creates a temporary (virtual) queue of machine-task mappings by using the PET and convolving it with the PTC from the currently assigned tasks to machine's queue. These mapping events attempt to map batches of tasks from the batch queue. This happens until either the machine queues are full, or there are no more unmapped tasks. The machines use limited-size First Come First Serve (FCFS) local queues to process the assigned tasks. It is assumed that once a task is mapped to a machine, its data is transferred to that machine and it cannot be remapped due to data transfer overhead [14]. It is assumed that the execution time of each task is independent, relies only on its own data and does not communicate with any other task, meaning that each task executes in isolation on a machine, with no preemption and no multitasking [44, 45, 46].

Figure 3.2. Batch mode operates on batches of tasks. Immediate mode operates on a single task immediately.



3.3 Benchmark Resource Mapping Heuristics for Heterogeneous Distributed Systems

This section describes the implemented heuristics used to map arriving tasks to heterogeneous machines. Both homogeneous and heterogeneous scheduling heuristics are described. In general, mapping methods in a system with dynamic task arrival can operate in immediate (on-line) or batch modes [14]. However, it has been proven that batch mode mapping heuristics offer a better performance and improve robustness in compare with immediate mode heuristics [14, 25]. Accordingly, batch mode heuristics are studied for the majority of this study and the proposed method is compared against them.

3.3.1 Heterogeneous batch mode heuristics. Four benchmark batch mode mapping heuristics are examined in this work. These mapping heuristics operate on batches of tasks, as shown in Figure 3.2. Each of these heuristics is a two-phase process, where unmapped tasks are copied into a virtual queue, a first phase finds the best machine for each task, by virtue of a per-heuristic objective. In the second phase, from task-machine pairs obtained in the first phase, each heuristic attempts to choose the

best machine-task pairs for each available machine queue slot. After all slots are filled, or when the unmapped queue is emptied, the virtual mappings are pushed (assigned) to the machine queues, and the mapping method is complete.

3.3.1.1 MinCompletion-MinCompletion (MM). This heuristic has seen much use and examination in the literature [47, 48, 49, 14], and serves as a baseline for the tests. The PET matrix is used to calculate expected completion times. In the first phase of this two-stage heuristic, the virtual queue is traversed, and for each task in that queue, the machine with the minimum expected completion time is found, and a pair is made. In the second phase, for each machine with a slot in its machine queue, the provisional mapping pairs are examined to find the machine-task pair with the minimum completion time, and the assignment is made to the virtual machine queues. The process repeats itself until all virtual machine queues are full, or until the temporary batch queue is exhausted.

3.3.1.2 MinCompletion-Soonest Deadline (MSD). This heuristic is a two-phase process, first selecting the machine which provides the minimum expected completion time (using the PET matrix) for each task under consideration. From this list of possible machine-task pairs, the tasks for each machine with the soonest deadline are chosen, and in the event of a tie, the task with the minimum expected completion time breaks the tie. As with MM, after each machine with an available virtual queue slot receives a task from the provisional mapping in phase two, the process is repeated until either the virtual machine queues are full, or the temporary unmapped task queue is empty.

3.3.1.3 MinCompletion-MaxUrgency (MMU). MMU is a two-phase process, the second of which is focused on urgency. Urgency for task i on machine j is defined as the inverse of the difference between the task deadline (δ_i) and the expected completion time of the task on machine j ($\mathbf{E}[C(t_{ij})]$). Equation 3.6 formally shows the urgency definition.

$$U_{ij} = \frac{1}{\delta_i - \mathbf{E}[C(t_{ij})]} \tag{3.6}$$

As with the previous heuristics, phase one finds the minimum expected completion time machine (using PET matrix) for each task and makes a task-machine pair. Using the urgency equation, phase two selects the task-machine pair that has the greatest urgency, and adds that mapping to the virtual queue. The process is repeated until either the temporary batch queue is empty, or until the virtual machine queues are full.

3.3.1.4 Max On-time Completions (MOC). The MOC heuristic is based on the algorithm proposed in [14] and uses the PET matrix to calculate completion times. It is also a two-phase heuristic, with a culling step in between, based on finding the robustness of task-machine mappings. The first mapping phase finds, for each task, it finds the machine offering the highest robustness value. The culling phase clears the virtual queue of any tasks that fail to meet a given robustness threshold α. In [14], α is set to 30%. The last phase finds the three virtual mappings with the highest robustness and permutes them to find the task-machine pair that maximizes the overall robustness and maps it to that machine's virtual queue. The process repeats until either all tasks in the temporary batch queue are mapped or dropped, or until the virtual machine

queues are full.

3.3.2 Heterogeneous immediate mode heuristics. Five immediate mode mapping heuristics are used in this study. These mapping heuristics map tasks to machines as they arrive, as shown in Figure 3.2.

3.3.2.1 First Come First Served (FCFS). In FCFS the task is assigned to the first available machine. Machine 0 is checked, then Machine 1, etc... all the way to Machine n.

3.3.2.2 Min Execution Time (MET). In MET the task is assigned to the machine which offers the minimum expected execution time (*i.e.,* the average of the PET[i,j] PMF for task i on machine j)

3.3.2.3 Min Completion Time (MCT). In MCT the task is assigned to the machine which offers the minimum expected completion time by convolving the distributions of all the tasks on a given machine using the expected execution time (*i.e.,* the average of the PET[i,j] PMF for task i on machine j).

3.3.2.4 K-Percent Best (KPB). KPB is an optimization of MCT that only considers the MCT of the K percent of machines with the best expected execution times (*i.e.,* the average of the PET[i,j] PMF for task i on machine j) for a given task.

3.3.2.5 Max Robust (MR). MR is a heuristic proposed in [14] that uses the full potential of the PET to calculate the machine-task pairs with the best robustness from the k-percent machines with the best expected execution times(*i.e.,* the average of the PET[i,j] PMF for task i on machine j).

3.3.3 Homogeneous computing scheduling heuristics. Three heuristics are used to study homogeneous computing environments. With the exception of First Come First Served, they operate on batches of tasks, as shown in Figure 3.2.

3.3.3.1 First Come First Served (FCFS). In FCFS the task is assigned to the first available machine. Machine 0 is checked, then Machine 1, etc... all the way to Machine n.

3.3.3.2 Earliest Deadline First (EDF). EDF is functionally similar to MSD from the heterogeneous batch mode heuristics. Though all machines are homogeneous in performance, the queues of each machine will have a different completion time during a mapping event. The first phase of this heuristic finds the task-machine mappings offering the minimum completion time. The second phase finds the mapping pair with the soonest deadline, and maps the pair's task to the pair's machine.

3.3.3.3 Shortest Job First (SJF). SJF is functionally similar to MM from the heterogeneous batch mode heuristics. Though all machines are homogeneous in performance, the queues of each machine will have a difference completion time during a mapping event. Instead of the traditional understanding of a 'job' being the execution time of the task, the job is considered to be the completion time of the task when mapped to a queue. The first phase of this heuristic finds the task-machine mappings offering the minimum completion time. The second phase finds the mapping pair with the minimum completion time, and maps the pair's task to the pair's machine.

3.4 Evaluation Setup

3.4.1 Overview. In this study, to reduce simulation execution times, the number of machines comprising the distributed systems is constrained, but the proposed methods are scalable to any number of machines. For the experiments in this paper, the number of machines is eight (*i.e.,* $M = 8$). The eight machines comprise an inconsistently heterogeneous system where a given machine A can exhibit higher performance for certain task types than machine B, yet machine B may exhibit higher performance on other task types [7].

To generate the probabilistic execution time PMFs (PET), the mean execution time results from twelve SPECint benchmarks on a set of eight machines was determined [14]. These mean execution times for each benchmark on each system formed the mean values for the task-machine execution times. The function describing execution time of the tasks on a machine is assumed to be a unimodal distribution; from a gamma distribution using the task-machine mean execution time, and with a shape randomly picked from the range [1:20], 500 execution times were sampled. From these times, a histogram was generated to produce a discrete probability mass function (PMF) analogue to the original probability density function. This was repeated for each task type on each machine, and the resultant eight machine by twelve task-type matrix of PMFs was stored as the PET. The PET matrix is constant across all of the experiments.

3.4.2 Generating workload to evaluate mapping heuristics. The simulation is of a finite span of time units, starting and ending in a state where the

system is idle. As the system comes on-line, and tasks begin to accumulate in the queue, the system is not in the desired state of oversubscription. The same is true of the end of the simulation, when the last tasks are finishing, and no more are arriving to maintain the oversubscribed state. In an effort to minimize the effects of the non-oversubscribed portion of the simulation from the data, the first and last hundred (100) tasks to complete are removed from the results. Only the remaining tasks from the oversubscribed portion of the simulation are used in the analysis.

Based on previous workload investigations [14, 25, 23], a gamma distribution is created with a mean arrival rate for all task types that is synthesized by dividing the total number of arriving tasks by the number of task types. The variance of this distribution is 10% of the mean. Each task type's mean arrival rate is generated by dividing the number of time units by the estimated number of tasks of that type. A list of tasks with attendant types, arrivals times, and deadlines is generated by sampling each task type's distribution.

The task types each have a mean duration, from which the PET described above is derived. This mean duration is used in generating the deadlines for the tasks that arrive in the system. For a given task, as noted in Equation 3.7, the deadline is calculated by adding the mean duration for that task type (avg_i) to the arrival time (arr_i), and then adding in a slack period based on the mean of all task type's duration multiplied by a tuning parameter ($\beta \cdot avg_{all}$). This slack allows for the tasks to have a chance of completion in an oversubscribed system.

$$\delta_i = arr_i + avg_i + (\beta \cdot avg_{all}) \tag{3.7}$$

3.5 Summary

The system uses a queue of unmapped tasks and a mapping agent to map incoming tasks to the queues of available computing resources. These tasks have hard deadlines. The system is inconsistently heterogeneous (*i.e.,* some tasks perform better on some machines, others worse), and the system has more tasks than it can handle (*i.e.,* is oversubscribed). To maximize the number of tasks meeting their deadlines, the next chapter will introduce the Probabilistic Task Pruner.

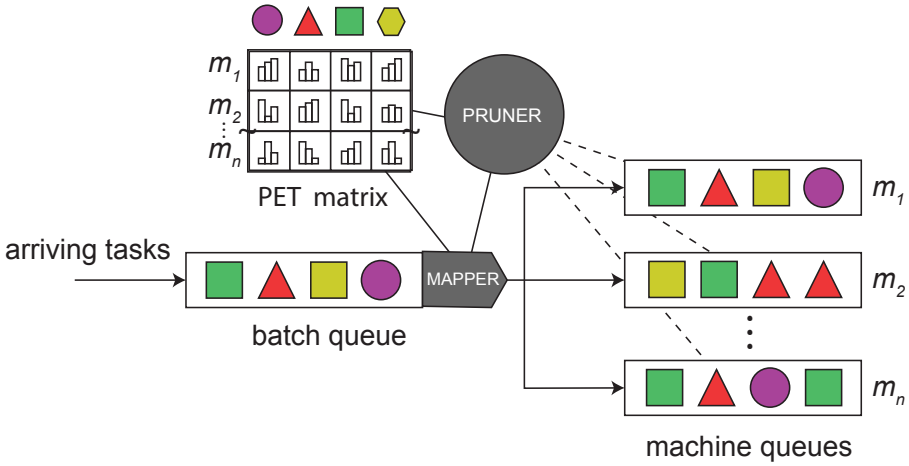
Chapter 4: Probabilistic Deferring and Dropping

4.1 Overview

In this chapter, a probabilistic task pruning mechanism (*i.e.,* Pruner) is developed and tested. As shown in Figure 4.1, this mechanism is used in conjunction with the mapping method to increase robustness of the system, either through deferring or dropping of tasks with low chance of success.

At each mapping event, before the mapping method is invoked, if the system is sufficiently oversubscribed, the Pruner comes to play and drops from machines the tasks with low chance of success. Then, the mapping method is invoked and determines the best mapping for tasks in the batch queue. Prior to assigning the tasks to machines, the tasks with low chance of success are deferred (*i.e.,* not assigned to machines) and returned to the batch queue to be considered during the next mapping events.

Figure 4.1. In each mapping event, the Pruner drops or defers tasks based on the tasks' probability of success.



4.2 Probabilistic Task Dropping

In an attempt to maximize the robustness of the system, the aggression of task pruning is dynamically adjusted in reaction to the level of oversubscription.

As the task arrival rate increases and the system becomes more oversubscribed, the probability of tasks missing their deadlines increases. As such, the number of tasks missed their deadlines since the past mapping event is used by the Pruner as an indicator to identify the level of oversubscription in the system. The identified level of oversubscription is used as a "toggle" that transitions the HC system to task dropping mode.

In the beginning of the mapping event, as shown in Algorithm 1, if the the system is identified as oversubscribed (step 1), the Pruner examines the machine queues: Beginning at the executing task (queue head), for each task in a queue, the success probability (*i.e.,* robustness) is calculated (step 4) based on its completion time PMF. Tasks whose robustness is less than or equal to the pruning threshold are dropped from the system. The proper values for toggle and pruning threshold is elaborated on in Section 4.4

In a given machine queue, as the Pruner drops tasks, the completion time PMF of those tasks behind the dropped tasks is improved. The way the completion time PMF of remaining tasks are changed as a result of dropping is explained in [14]. Intuitively, each task in queue compounds the uncertainty in the completion time of the tasks behind it in the queue. Dropping a task exclude its PET from the convolution process, hence, reduce the compound uncertainty. In addition to reduced uncertainty,

Algorithm 1: Algorithm for probabilistic task dropping.

Data: M is a set of machine queues $m_1, m_2, \ldots, m_q$
T is a set of tasks $t_1, t_2, \ldots, t_n$
d the oversubscription level indicator
$\mathbb{P}(t_i)$ is the success Probabilistic of task i
TOGGLE is the dropping toggle
THRESHOLD is the pruning threshold
Result: Algorithm to engage dropping

```
1 if d ≥ TOGGLE then
2     foreach machine queue m_j do
3         foreach task t_i on m_j do
4             if P(t_i) ≤ THRESHOLD then
5                 Drop t_i;
```

dropping tasks with low chance of success enables the tasks behind it to begin execution sooner, thus, increasing the overall robustness. As a matter of implementation, to reduce bottlenecks in PMF convolution calculation, it is likely that the pruning of machine queues would happen per machine, or perhaps per some subset of machines.

4.3 Pruning-Aware Mapping Method (PAM)

In each mapping event, after checking for probabilistic dropping, the mapping method is started. The main responsibility of mapping method is to find and assign tasks in the batch queue to the best available machine so that the overall robustness of the HC system is improved. In addition, the mapping method takes care of the "deferring" part of the pruning mechanism. Therefore, the mapping method is pruning-aware that means it makes use of pruning to improve the overall robustness of the HC system.

The basis for PAM comes from MOC, developed in [14]. The pseudo-code of PAM is shown in Algorithm 2. To find the best mapping, the mapper creates a

temporary *virtual queue* for all machines to perform its calculations. The virtual queue consists of tasks currently assigned to each machine and the tasks in batch queue. In the virtual queue, PAM drops all the tasks that have already missed their deadlines (steps 1 – 3 in Algorithm 2).

Much like MOC, in the first phase, the proposed mapping method (called *PAM*) consults with PET matrix and virtual queue (as explained in Chapter 3) to calculate completion time PMF (*i.e.,* PCT) for all tasks in the batch queue. Based on the obtained PCTs, for each task, the method can recognize the best probability of success (*i.e.,* task robustness) across all machines. This is shown as step 4 and 5 in Algorithm 2.

Differing from MOC, in the second phase, pruning occurs and defers mapping of tasks whose probability of success is lower than the pruning threshold (see steps 6 – 8). Essentially, this pruning ensures that no tasks will be scheduled that would be immediately dropped, in the case probabilistic dropping is triggered in the next mapping event. It is noteworthy that the pruned tasks are merely deferred, meaning that they are not considered in the current mapping event but remain in the unmapped queue to be considered in the next mapping event. Due to both inconsistent heterogeneity of HC systems and task dropping, it is likely that in the next mapping event, a deferred task can get a higher probability of success. This happens because those machines that provide a better performance for the deferred task can potentially become available by the next mapping event.

To manage the compound uncertainty, resulting from convolution of several tasks assigned to the same machine, the size of local machine queues in the HC system

is limited. Therefore, in the third phase of PAM, for each machine with room in its machine queue, from the remaining task mappings in the virtual queue, the task with the minimum completion time is selected and is assigned to the paired machine (see steps 9 – 12). The task is removed from the system batch queue, and the process repeats, until either the virtual queue is empty, or the machine queues are full.

Algorithm 2: Pseudo-code for Pruning-Aware Mapping Method (PAM)

```
   Data: Tasks in batch queue are copied to a virtual queue
   δ_i represents the deadline of task t_i
   THRESHOLD is the pruning threshold
 1 foreach task t_i in virtual queue do
 2     if δ_i < current time then
 3         Drop t_i;
 4     Find m_j offers the highest success probability;
 5     Add t_i to candidate tasks list for m_j as t_ij;
 6 foreach task-machine pair t_ij ∈ list do
 7     if ℙ(t_ij) ≤ THRESHOLD then
 8         Defer t_ij (return it to batch queue);
 9 foreach machine m_j ∈ m do
10     if m_j < maximum queue size then
11         Find t_i from list with minimum completion time on m_j;
12         Assign t_i to m_j;
```

4.4 Probabilistic Pruning Performance Evaluation

4.4.1 Experimental overview. A series of job simulations were run using the Louisiana Optical Network Infrastructure (LONI) Queen Bee 2 HPC system [10]. For each set of tests, for each examined parameter, 30 workload trials are performed using different task arrival times built from the same arrival rate and pattern, and the mean and 95% confidence interval of the results is examined.

PAM, and four baseline batch mode mapping heuristics are evaluated against

each workload trial to study how they behave when the parameters of the system vary. In particular, study focuses on the impact of varying: (A) pruning threshold (*i.e.,* the probability, under which a task is pruned); (B) the dropping toggle (*i.e.,* the oversubscription level, after which the system transitions to a more aggressive mode and engages probabilistic dropping of tasks); (C) the level of oversubscription (the task arrival rate). In addition to evaluating the influence of these parameters, the impact of the awareness of the pruning threshold during task mapping is studied. That is, the performance of PAM when stripped of awareness of the pruning threshold is studied, as well as the performance MOC when given awareness. The isolated impact of both parts of the pruning mechanism are also tested (deferring and dropping tasks).

Each experiment is a set of 30 workload trials, consisting of 800 tasks per trial. Each of the experiments investigates heavy levels of oversubscription (an arrival rate of 19k tasks per period) where very few tasks complete successfully using scalar expectation-based heuristics, such as MM, MMU, and MSD. Each machine in the HC system has a machine-queue size of six (6), counting the executing task. For each of the five mapping heuristics examined, the overall robustness of the system is calculated, which is the percentage of tasks completed before their deadline (the vertical axis in the figures of this section).

4.4.2 Impact of varying pruning threshold. To examine the impact of the pruning threshold in oversubscribed systems, performance is evaluated at 5% increments (from 0% to 100%), and confidence intervals are shown at 95%. In this experiment, the dropping toggle is set to a value of 1 missed task.

Figure 4.2. Percentage of tasks completed on-time (vertical axis) examining the effects of different pruning thresholds (horizontal axis) for pruning and dropping tasks. Dropping toggle is set to 1 missed task. 95% confidence intervals are shown.

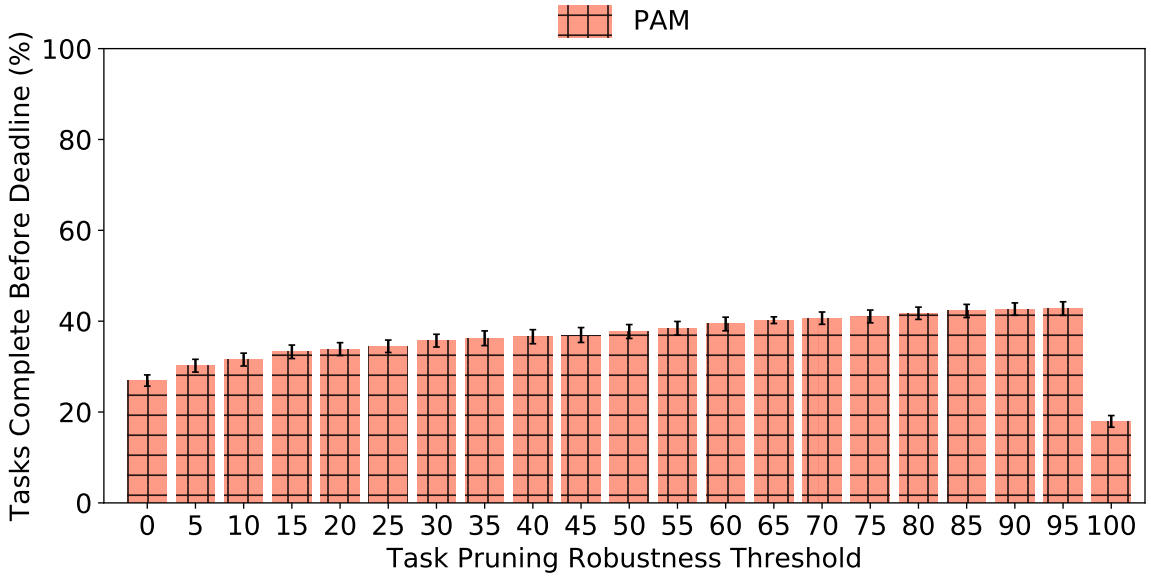


Figure 4.2 shows that the worst performance comes when tasks with 100% chance of success are dropped, meaning that once the pruner is engaged to drop tasks, all tasks are flushed from the system. This is no surprise. The second worst performance comes at a pruning threshold of 0%, which is when there is no probabilistic dropping, and tasks are only dropped after their deadlines have passed. Between these two extremes, the number of tasks meeting their deadline steadily increases as the dropping threshold increases. This is because of the extreme level of oversubscription in the system, and the nature of the task-machine PMFs, which result in tasks that possess either high or low probabilities of success, but with few probabilities that lie in between.

Based on these observations, it can be concluded that, in an oversubscribed HC system tasks should be mapped (scheduled) to machines, only if there is a high

confidence (75%) about the success chance of the tasks. As stated in the introduction, one of the goals in this thesis is to know a suitable probability of pruning threshold. While the mean of the results are different between 75% and 95%, the result is not statistically significant with the sample size tested. Because of this, 75% is chosen as a suitable probability, and PAM is configured with the a pruning threshold of 75% for the rest of experiments.

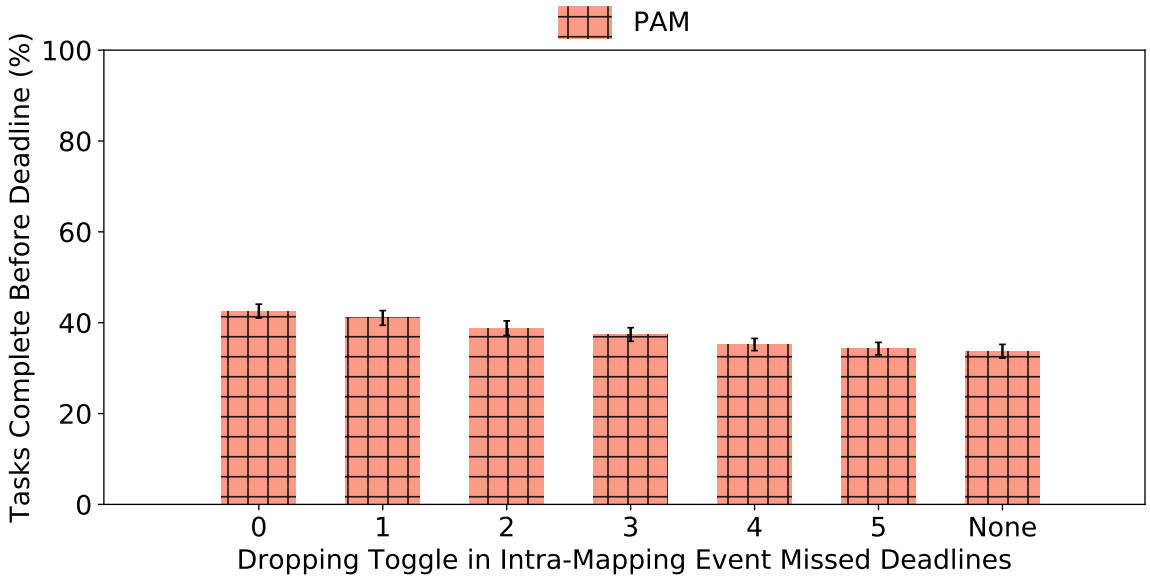
4.4.3 Impact of different toggles to engage task dropping. As stated in the introduction, to maximize the overall robustness of the HC system, a suitable oversubscription level under which the pruning should transition from only task deferring to a more aggressive mode, task dropping must be found. This suitable oversubscription level is called a "toggle". The value of the toggle shows how conservative the pruning mechanism should be in dropping tasks. That is, a lower toggle value of indicates that task dropping should be engaged as soon as a task is observed missing its deadline, whereas a higher toggle value means that the task dropping should not be engaged, unless a sufficient number of tasks are observed missing their deadlines (*i.e.,* higher level of oversubscription).

Accordingly this experiment examines the impact of engaging the task dropping mechanism at different toggle values and find the toggle value that maximizes the robustness of the HC system in an oversubscribed scenario.

The result of this experiment is shown in Figure 4.3. The horizontal axis shows the different toggle values evaluated. A toggle value of 0 means the Pruner engages task dropping always, regardless of the system being oversubscribed. The next values of

toggle indicate engaging task dropping in a more conservative manner. Specifically, the right-most toggle indicates never engaging probabilistic dropping (concisely labeled "None" in Figure 4.3). The vertical axis shows the robustness of the HC system via the percentage of tasks meeting their deadlines. In this experiment, the pruning threshold is 75%, and the total number of tasks in the trials is 800.

Figure 4.3. Percentage of tasks completed on-time (vertical axis) examining the effects of different dropping toggles (horizontal axis) to engage task dropping. Pruning threshold is set to 75%, and the arrival rate is 19k.



In Figure 4.3, there is a steady increase in performance between no probabilistic dropping, and a toggle of 0 (the least conservative dropping strategy). As the pruning aggression increases, as the toggle moves toward zero, PAM perform better, from 34% tasks completed on time to 43%. This increase in performance is because the dropping of unlikely to succeed tasks (less than 75% probability of success) frees up processing resources, allowing them to be more effectively allocated to tasks with better affinity.

PAM, in particular, benefits from knowledge of the pruning threshold and does

not schedule tasks that are guaranteed to be dropped during the next mapping event. Whereas, when faced with oversubscription sufficient to engage dropping, the MOC heuristic can potentially map tasks that are guaranteed to be dropped during the next mapping event.

4.4.3.1 Impact of pruning threshold awareness. In the previous part, a suitable pruning threshold has been identified. In this experiment, the intention is to analyze the impact of the scheduler's awareness of the dropping threshold. The goal is to separate the robustness-based logic of PAM from its awareness of the dropping threshold when making the decision to defer or map tasks. Accordingly, the analysis has two parts: first, comparing the PAM logic against another robustness based mapping heuristic; second, comparing PAM with a variation of itself that is unaware of pruning.

For the first part, PAM is evaluated against MOC which is the other robustness-based batch-mode mapping heuristic implemented in this study. To concentrate the analysis on the logic of the heuristic, MOC is made aware of the suitable dropping threshold. That is, because the dropping threshold is 75%, tasks which are less likely than that to succeed will not be scheduled. This aware version of MOC will be called MOCA. For the second part, since the pruning mechanism of PAM depends on two uses of probabilities, namely deferring threshold and dropping threshold. To evaluate the impact of these parts, a variation of PAM that is unaware of the deferring threshold is created, which uses a predefined value of 30% (the same as MOC). This unaware version of PAM is known as Pruning Unaware Mapping (PUM).

The result of this experiment is illustrated in Figure 4.4. The horizontal axis of

the experiment shows the four mapping methods evaluated and the vertical axis shows the robustness of the HC system in form of percentage of tasks meeting their deadlines. The rest of experimental set up is similar to Section 4.4.2

Figure 4.4. Percentage of tasks meeting their deadlines (vertical axis) analyzing the impact of pruning threshold awareness on robustness-based mapping heuristics. The pruning dropping toggle is set to 1 missed task.

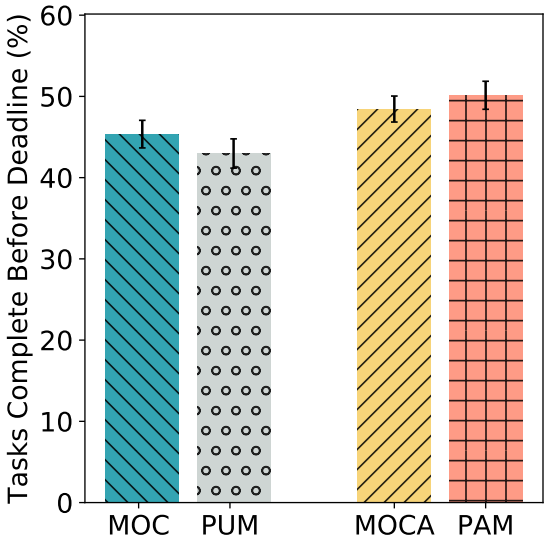


Figure 4.4 shows that integrating awareness of the pruning threshold with the mapping heuristics allows MOCA to reach parity with PAM, and to map more tasks that meet their deadlines than MOC. As for MOC, even as it is a more complicated heuristic that calculates many permutations of task-machine pairs to find the most robust mapping, the benefits of awareness of the pruning threshold are significant. For PAM, which is simpler in its calculations than MOC, the benefits of awareness of the dropping threshold are even more pronounced, as PUM results in slightly fewer successful tasks than MOC in the trials. This means that in some cases of extreme oversubscription, simpler heuristics can outperform more complicated ones when there

is an awareness of the conditions required for a task's success in the system.

4.4.4 Impact of dropping without deferring. To examine the impact of the pruning threshold in oversubscribed systems, performance is evaluated with deferring at 75% required robustness, and without deferring, and confidence intervals are shown at 95%. In this experiment, when dropping, the dropping toggle is set to a value of 1 missed task. A high (19k) and an extreme (34k) level of oversubscription are tested.

Figure 4.5. Percentage of tasks completed on-time (vertical axis) examining the effects probabilistically dropping tasks, isolated from deferring. The horizontal axis is the level of oversubscription in tasks per period. Dropping toggle is set to 1 missed task, when enabled. 95% confidence intervals are shown.

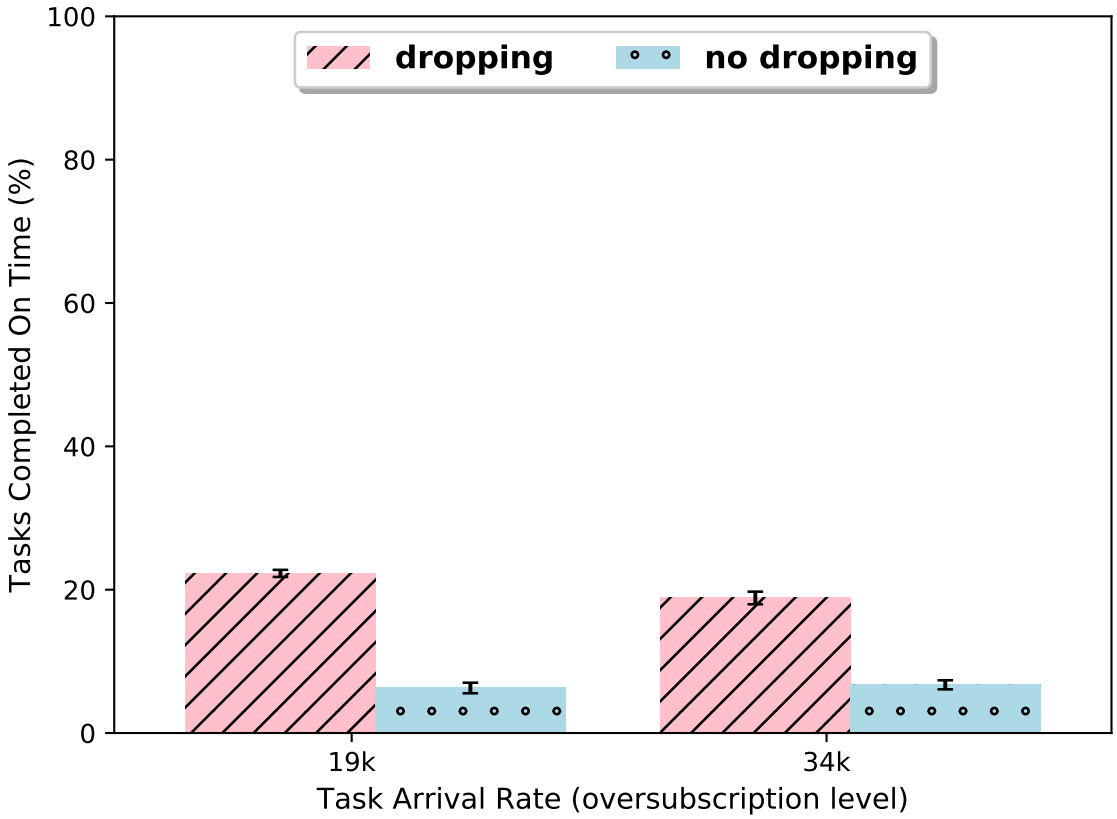


Figure 4.5 shows that probabilistically dropping tasks will, alone, be effective in

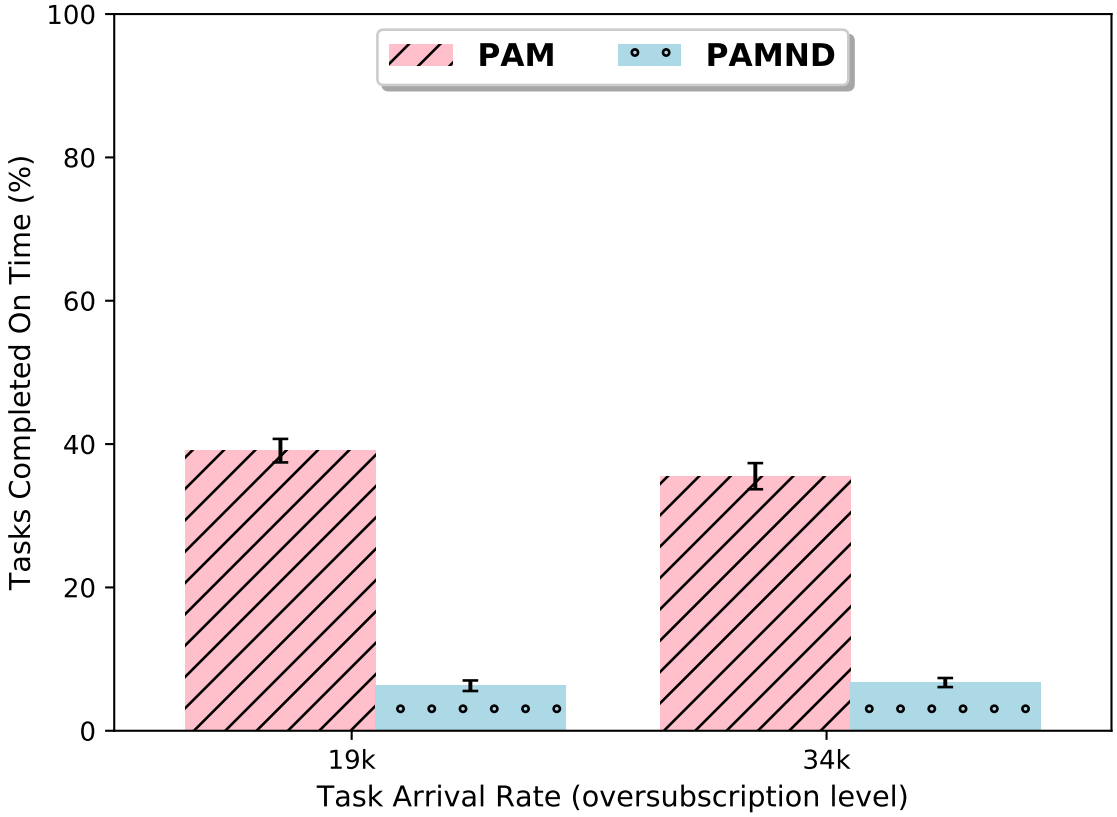
increasing the robustness of HC systems under heavy oversubscription. There is about a fourfold increase in the percentage of tasks completing before their deadlines. This holds true even at extreme levels of oversubscription, with only a slight reduction in overall performance. This is attributable to the ability to recover from poor mapping decisions that probabilistic task dropping offers to a system. Once a mapping has proven insufficient, it is dropped, allowing resources to be used by other tasks, thereby increasing system robustness.

4.4.5 Impact of deferring without dropping. To examine the impact of the isolated deferring portion of the pruning mechanism in oversubscribed systems, performance is evaluated with deferring and without, and confidence intervals are shown at 95%. A high (19k) and an extreme (34k) level of oversubscription are tested.

Figure 4.6 shows that probabilistically deferring tasks has nearly double the impact on system robustness than probabilistically dropping. Nearly 40% of tasks complete on time at heavy levels of oversubscription while deferring, compared to nearly 6% without. This makes sense, as deferring allows the system to await resources that are likely to successfully complete tasks on time, as opposed to prematurely scheduling the best available fit. The performance of deferring compared to dropping suggests that it is of greatest benefit to prevent bad mappings, but when unavoidable, or when circumstances change, responding to bad mappings will also help maximize system robustness.

4.4.6 Impact of level of oversubscription. As a suitable pruning threshold and the proper toggle value have been found in the previous experiments, it remains

Figure 4.6. Percentage of tasks completed on-time (vertical axis) examining the effects of probabilistic deferring isolated from task dropping. The horizontal axis is the level of oversubscription in tasks per period. PAMND has no dropping enabled. 95% confidence intervals are shown.



necessary to verify the performance of PAM (using those identified configurations) under varying oversubscription levels. An additional goal is to learn how PAM performs with respect to other mapping heuristics under lighter and heavier oversubscription levels. Accordingly, this experiment examines the effect of oversubscription level on system robustness when different mapping heuristics are in place. For this experiment, the pruning threshold is set to 75%, and the dropping toggle is set to 1.

In this experiment, shown in Figure 4.7, the horizontal axis shows the arrival

rate of tasks. Higher arrival rates indicate higher level of oversubscription. As 19k was the oversubscription level of other experiments, a lower level of oversubscription (9k and 14k) and two higher oversubscription levels (24k, 29k, and 34k) are evaluated herein. The lower bound of 9k is chosen because of the nearly 100% of on-time completions. The vertical axis shows the robustness of the system in terms of number of tasks meeting their deadlines within a time-frame. This is different than the previous use of completion percentage, as the number of tasks better shows the effects of heuristics under varying workloads.

Figure 4.7. Number of tasks meeting their deadlines (vertical axis) examining the impact of different task arrival rates. Dropping toggle is set to 1 missed task. Pruning threshold is set to 75%. The horizontal axis is the level of oversubscription in tasks per period.

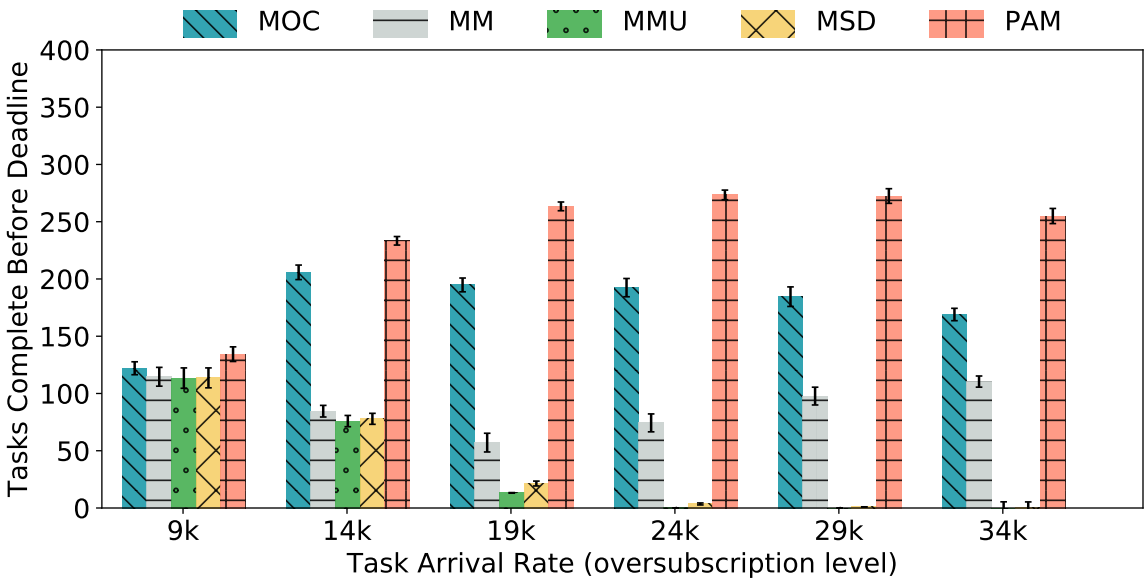


Figure 4.7 shows that when there is little-to-no oversubscription, while PAM does perform better than the other tested heuristics, the difference is not that great. As the level of oversubscription increases, heuristics that attempt to minimize deadline

misses (MSD and MMU) begin to perform drastically worse, whereas heuristics that attempt to maximize on-time completions or minimize completion time (PAM, MOC, and MM) do not suffer the same fate of zero on-time completions at high levels of oversubscription. As the oversubscription rises, MM, MOC, and PAM each find their steady state of on-time completions, and their performance is in that order. PAM outperforms MOC by 75%, and doubles the performance of MM. This is due to PAMs ability to defer mapping tasks that are likely to be dropped when faced with higher levels of oversubscription, coupled with aggressively dropping tasks that are unlikely to succeed, allowing for those tasks, most likely to succeed, to execute, resulting in a maximized system robustness.

4.5 Summary

This chapter shows that for a given oversubscribed heterogeneous system there is a probability at which a task is not worth executing. It shows that by deferring the mapping of tasks who are estimated to be below that threshold, or by dropping queued or executing tasks that have dropped below that threshold, the overall number of tasks to successfully complete in a system goes up. It shows this by creating and testing a Pruning Aware Mapper (PAM), and shown its effectiveness versus other benchmark heuristics at a variety of levels of oversubscription. In the next chapter, the benefits of applying the deferring and dropping mechanisms from the Pruner to existing batch and immediate mode heuristics are examined.

Chapter 5: Applying Probabilistic Pruning to Existing Mapping Heuristics

In the previous chapter, a Pruning Aware Mechanism PAM is developed and tested to show the effectiveness of probabilistic pruning in increasing the robustness of heterogeneous computing systems. The focus of this chapter is examining the effectiveness of pruning as a plug-in addition to existing scheduling heuristics. The first experiments are concerned with the batch mode heuristics used in Chapter 4. The final experiment examines the effect of probabilistic dropping on simpler immediate mode heuristics.

5.1 Overview

Probabilistic pruning has, as yet, been tested in the guise of PAM, the Pruning Aware Mapper. While new and improved mapping methods are important, perhaps the idea of replacing the scheduling heart of an HC system completely is an idea that will fail to gain traction. To that end, the components that make up PAM, the deferring and the dropping portions of the probabilistic task pruner should be proven effective in improving the robustness of a variety of currently used mapping heuristics. Most of these systems use batch mode heuristics (described in Section 3.3). Some, however use immediate mode heuristics (described in Section 3.3), and these shall also be examined. As a final point in this chapter, a series of tests are run on homogeneous distributed computing scheduling heuristics to investigate the benefits of probabilistic pruning therein. The heuristics are described in 3.3.3

5.1.1 Experimental overview. A series of job simulations were run using the Louisiana Optical Network Infrastructure (LONI) Queen Bee 2 HPC system [10]. For

each set of tests, for each examined parameter, 30 workload trials are performed using different task arrival times built from the same arrival rate and pattern, and the mean and 95% confidence interval of the results is examined.

Three conventional baseline batch mode mapping heuristics are evaluated against each workload trial to study how they behave when equipped with part of, or all of, probabilistic pruning. When equipped with deferring, the heuristic is noted with a "-D", when equipped with all of pruning, it is noted with a "-P". Of particular interest is the impact of: (A) probabilistic deferring only; (B) probabilistic dropping only; (C) both dropping and deferring tasks probabilistically. Each of these experiments is tested at different levels of oversubscription (the task arrival rate).

Each experiment is a set of 30 workload trials, consisting of 800 tasks per trial. Each of the experiments investigates extreme levels of oversubscription (an arrival rate of 19k tasks per period) where very few tasks complete successfully using scalar expectation-based heuristics without pruning, such as MM, MMU, and MSD. Each machine in the HC system has a machine-queue size of six (6), counting the executing task. For each of the mapping heuristics examined, the overall system robustness is calculated, which is the percentage of tasks completed before their deadline (the vertical axis in the figures of this section).

5.2 Performance Evaluation of Batch Mode Heuristics

As in the previous chapter, at each mapping event, before the mapping heuristic is invoked, if the system is sufficiently oversubscribed, the Pruner comes to play and drops from machines the tasks with low chance of success. Then, the heuristic is

invoked and determines the best mapping for tasks in the batch queue. Prior to assigning the tasks to machines, the tasks with low chance of success are deferred (*i.e.,* not assigned to machines) and returned to the batch queue to be considered during the next mapping events. The heuristics used are explained in Section 3.3.1

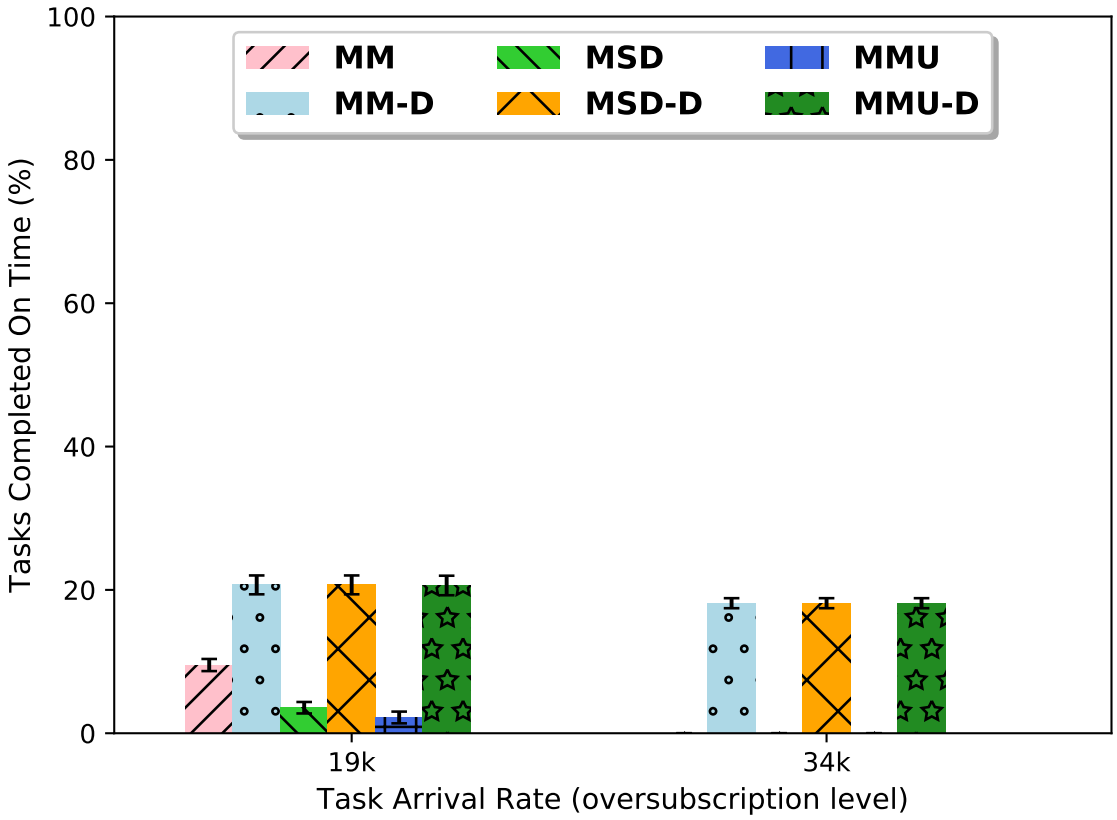
5.2.1 Effect of probabilistic deferring on system robustness. This experiment examines the effect of adding probabilistic deferring to batch mode heuristics on system robustness at varying oversubscription levels. For this experiment there is no probabilistic dropping, only deferring, or not.

The vertical axis shows the robustness of the system in terms of percentage of tasks meeting their deadlines.

Figure 5.1 shows that at high levels of oversubscription (the left collection of bars), deferring increasing the robustness of systems using MM, MSD, and MMU from single digits to more than 20%. These same heuristics at extreme levels of oversubscription (the right collection of bars) are unable to map any tasks to completion, but when deferring is added, are able to attain nearly 20% robustness in the face of such oversubscription. This is accountable to system deferring tasks until machines with high enough affinity become available to complete a task. By preventing the use of machines for tasks they are unsuited to, they are kept ready to execute tasks that they are suited to, thereby increasing the robustness of the system as a whole.

5.2.2 Effect of probabilistic dropping on system robustness. This experiment examines the effect of adding probabilistic dropping to batch mode heuristics on system robustness at varying oversubscription levels. For this experiment,

Figure 5.1. Percentage of tasks meeting their deadlines (vertical axis) examining the impact of deferring tasks. The horizontal axis is the level of oversubscription in tasks per period.

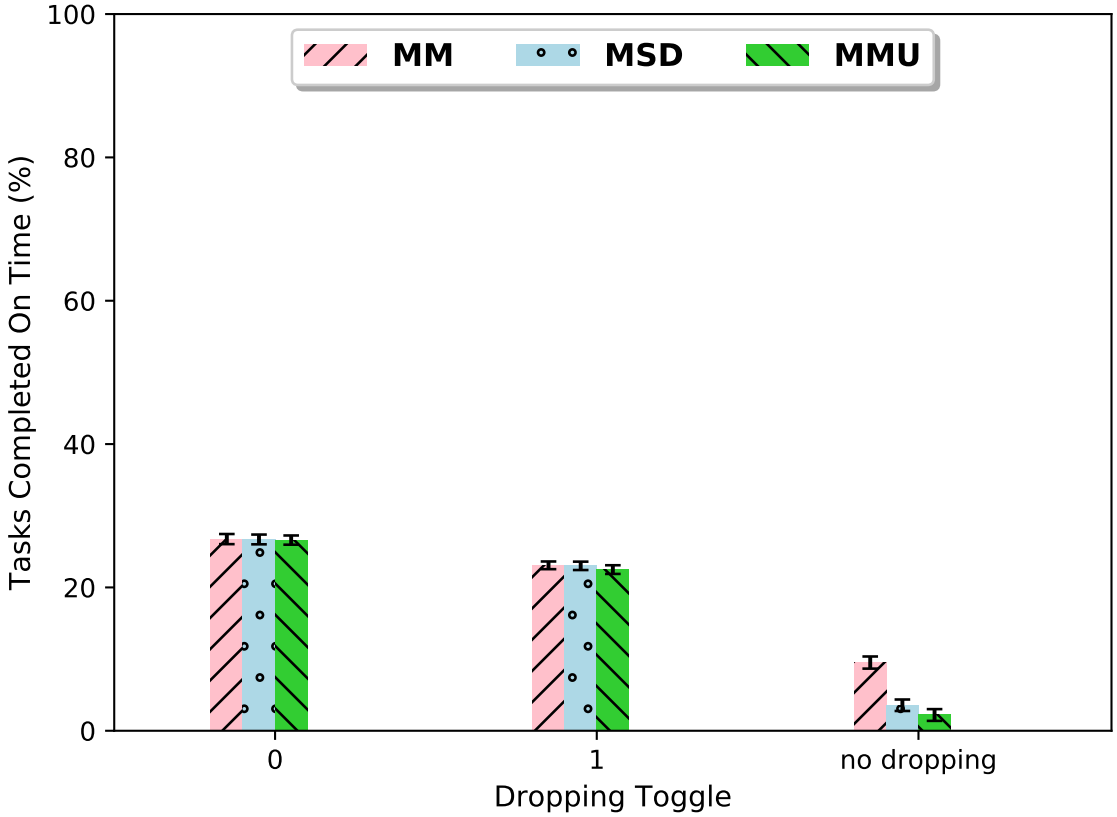


the pruning threshold is set to 75%, and the dropping toggle is set to 1.

The vertical axis shows the robustness of the system in terms of percentage of tasks meeting their deadlines.

Figure 5.2 shows an increase in the number of tasks completing before their deadline when using probabilistic task dropping. To engage dropping after a single dropped task results in more than quadrupling the robustness of the system in conventional task mapping heuristics such as MM, MSD, and MMU. Giving a system the ability to adjust to changing circumstances results in powerful increases in

Figure 5.2. Percentage of tasks meeting their deadlines (vertical axis) examining the impact of dropping tasks. Pruning threshold is set to 75%. Horizontal axis shows the dropping toggle in terms of the number of tasks missing their deadline between mapping events needed to engage pruning.



robustness. Finally, when engaging the dropping system at every mapping event, the overall system robustness increases even more, compared to only engaging after a task misses its deadline. This is accountable to the effective forgiveness for low-affinity mappings that task-dropping provides, by removing low-probability tasks from the queues and processors, allowing more likely tasks access to resources, increasing their chance of success.

5.2.3 Effect of probabilistic pruning on system robustness. This experiment examines the effect of adding both probabilistic deferring and dropping to batch mode heuristics on system robustness at varying oversubscription levels. For this experiment, the pruning threshold is set to 75%, and the dropping toggle is set to 1.

The vertical axis shows the robustness of the system in terms of percentage of tasks meeting their deadlines.

Figure 5.3. Percentage of tasks meeting their deadlines (vertical axis) examining the impact of task pruning (deferring and dropping). Dropping toggle is set to 1 missed task. Pruning threshold is set to 75%. Horizontal axis shows the dropping toggle in terms of the number of tasks missing their deadline between mapping events needed to engage pruning.

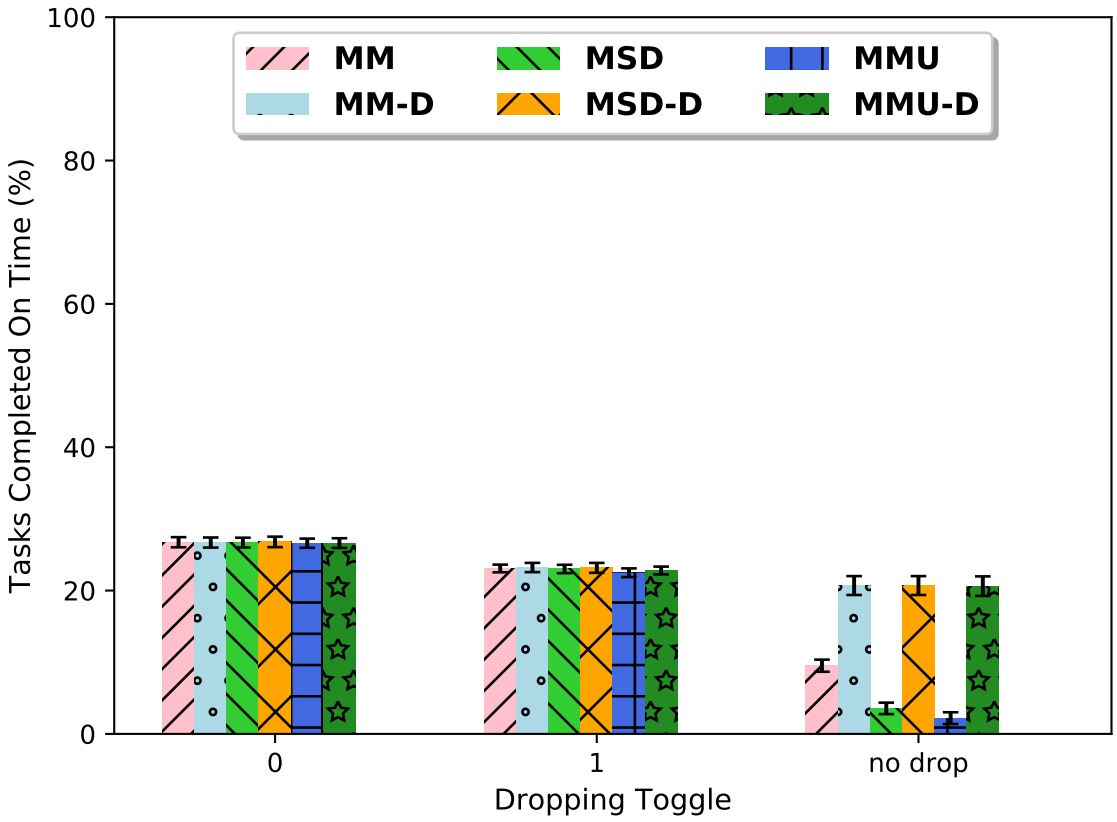


Figure 5.3 shows that when applying both probabilistic dropping and deferring

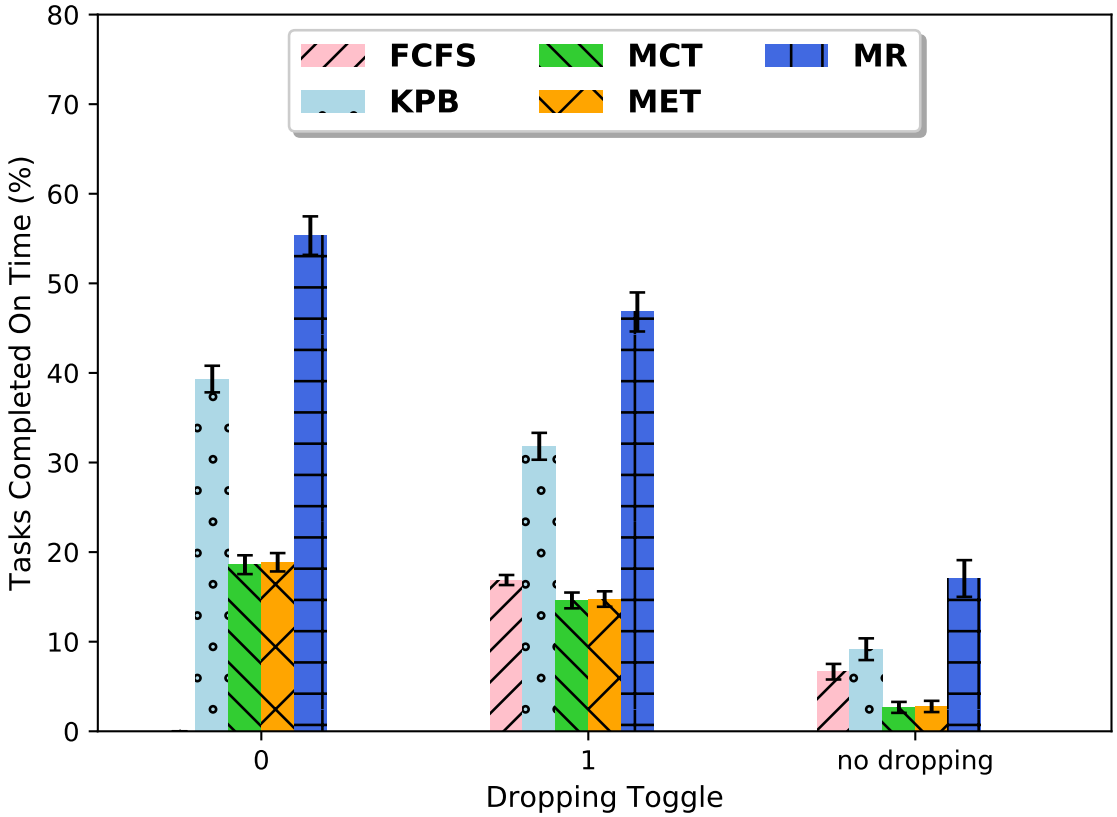
to existing heuristics, the best results in this oversubscribed system happen with using any of the heuristics with deferring and dropping at a toggle of 0 missed tasks. For all heuristics, both dropping and deferring together result in more successful tasks then applying either one, or the other, or none. This is a result of both waiting for machines that offer a high probability of success, and then reacting to the changing situation as the effects of oversubscription and compound uncertainty play out. The largest impact for heuristics such as MSD and MMU, however, comes from deferring tasks, as these heuristics attempt to map the tasks with the least slack between mapping time and the task's deadline, tending to a string of unlikely-to-succeed tasks being mapped. By ensuring a strict threshold, these tasks are not mapped, raising the overall system robustness.

5.3 Performance Evaluation of Immediate Mode Heuristics

In the literature, batch mode mapping heuristics are preferred due to the increased robustness of the mappings they make, however there do exist scenarios where immediate mode heuristics are used [6]. To improve the robustness of these systems, a dropping mechanism can be added. For each of the tested immediate mode heuristics, as tasks arrive they are immediately mapped to a machine in the system. The machine queues are essentially infinite. The heuristics used are explained in Section 3.3.2

Figure 5.4 shows that applying probabilistic dropping to immediate mode heuristics results in increased system robustness. When dropping is engaged at every mapping event, with such a high probability required for dropping, FCFS suffers because the mapping it produces is of too low a robustness to survive the pruning. The

Figure 5.4. Percentage of tasks meeting their deadlines (vertical axis) examining the impact of task dropping on immediate mode heuristics. Dropping toggle is set to 1 missed task. Dropping threshold is set to 75%. Horizontal axis shows the dropping toggle in terms of the number of tasks missing their deadline between mapping events needed to engage pruning.



more intelligent mapping methods (MR, KPB) showed the most benefit, as they were less likely to perform low-probability task-machine mappings, resulting in a higher percentage of tasks completing before their deadline. This suggests that mapping mechanism with high confidence can benefit from a high pruning threshold, and lower confidence mapping methods can benefit from a looser threshold.

5.4 Performance Evaluation of Mapping Heuristics in Homogeneous Computing Distributed Systems

The focus of this thesis is heterogeneous distributed computing systems, however many homogeneous distributed systems are in use, and it bears investigating the impact of probabilistic pruning on them. In an attempt to improve the robustness of these systems, both a deferring and a dropping mechanism can be added. The heuristics used are explained in Section 3.3.3

Figure 5.5. Percentage of tasks meeting their deadlines (vertical axis) examining the impact of task pruning on homogeneous computing scheduling heuristics. Dropping toggle is set to 1 missed task. Dropping threshold is set to 75%. The horizontal axis is the level of oversubscription in tasks per period.

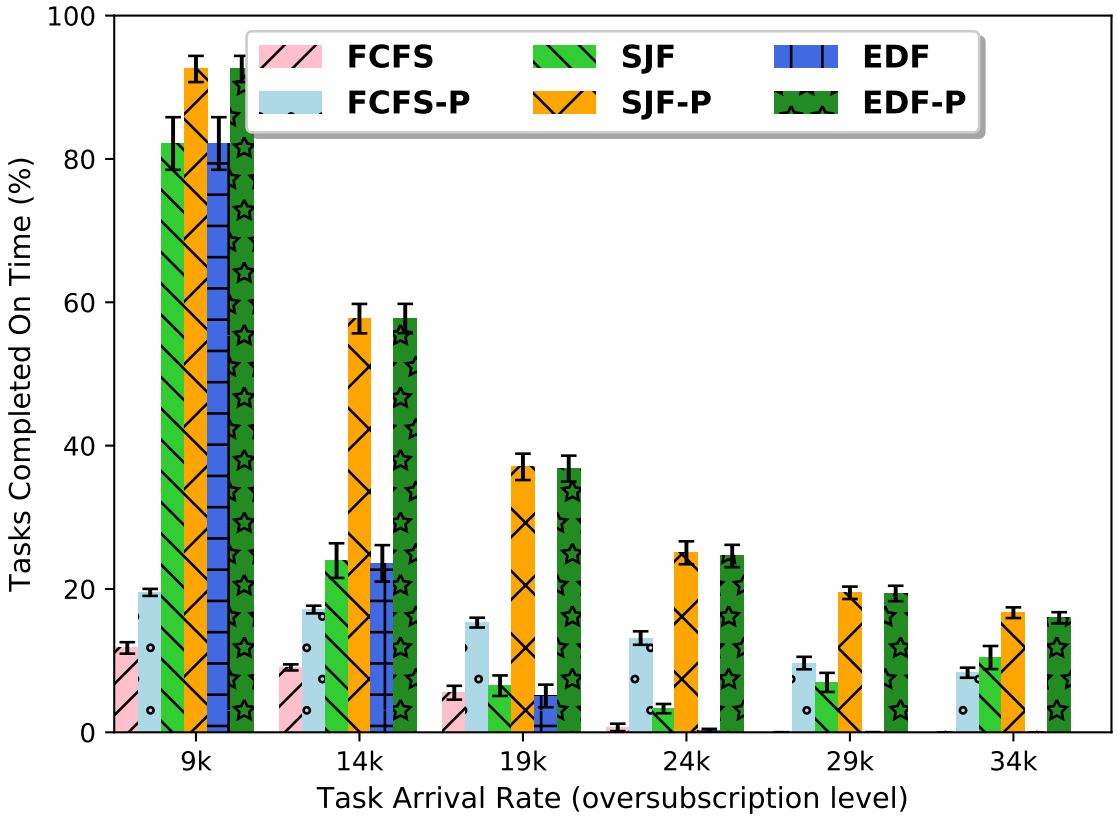


Figure 5.5 shows that applying probabilistic dropping to homogeneous systems results in increased system robustness at all levels of oversubscription. As the level of oversubscription increases, the gap in performance increases. At extreme levels of oversubscription, where EDF produces no on-time completions and SJF produces 14%, adding pruning can result in 18% and 20% on-time completions, respectively. This is because, as in heterogeneous systems, the pruning mechanism allows the system to recover from mapping tasks are are unlikely to successfully complete on time.

5.5 Summary

In this section the pruning mechanism developed in the previous chapter was applied, in pieces and in whole, to existing batch mode heuristics. Immediate mode heuristics were also tested with probabilistic dropping, showing an increase in system robustness. The benefits of probabilistic deferring and dropping can be bolted on to existing heuristics, thereby maximizing the number of successful tasks in the oversubscribed system as a whole. In the next chapter, ways to increase the effectiveness of PAM are developed, such as decoupling the dropping and deferring thresholds in the pruner, and other options and facets of the system are explored, such as fairness amongst task types and cost considerations.

Chapter 6: Advanced Pruning: Decoupling Dropping and Deferring

While the performance of probabilistic pruning has been shown to be effective, the method operates in a static, system-wide fashion. For every task in the system, in each case, the same probability is used to make decisions. Whether it is to defer a task from mapping, or to drop a currently running, or queued task, one probability is used for comparison.

The conditions to engage dropping (*i.e.,* passing a certain discreet threshold of dropped tasks between mapping events) is only one possible method. In this chapter, a schmitt-trigger like mechanism for engaging task-dropping is explored, as is the effect of changing the weight of missed-deadline history in engaging the dropping mechanism.

The following are examined:

- Decouple the probability used to defer mapping tasks from that used to drop tasks.
- Test the effectiveness of a per-task, dynamic, pruning threshold.
- Examine less-sensitive ways to engage the Pruner.
- Attempt to mitigate unfairness to task-types caused by the Pruner.
- Explore the costs of using PAM vs other heuristics

6.1 Experimental Overview

A series of job simulations were run using the Louisiana Optical Network Infrastructure (LONI) Queen Bee 2 HPC system [10]. For each set of tests, for each

examined parameter, 30 workload trials are performed using different task arrival times built from the same arrival rate and pattern, and the mean and 95% confidence interval of the results is examined.

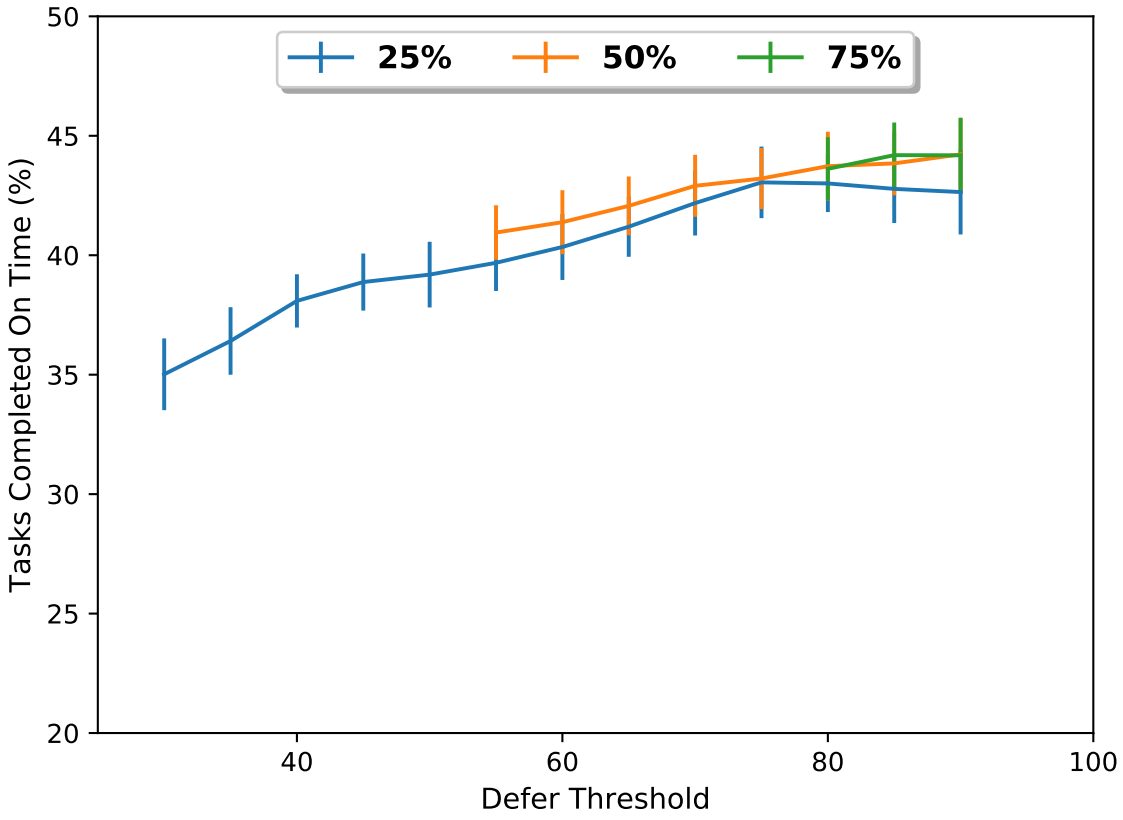
Each experiment is a set of 30 workload trials, consisting of 800 tasks per trial. Each of the experiments investigates extreme levels of oversubscription where very few tasks complete successfully using conventional scalar expectation-based heuristics, such as MM, MMU, and MSD. Each machine in the HC system has a machine-queue size of six (6), counting the executing task. For each of the experiments, unless otherwise noted, the performance metric is the overall robustness of the system, which is the percentage of tasks completed before their deadline (the vertical axis in the figures of this section).

6.2 Decoupled Deferring and Dropping

For the Pruner introduced in Chapter 4, a single threshold is used to decide whether to map a task with a system, or to defer, as well as to decide whether to drop a mapped task from its machine (*i.e.,* deferring and dropping). Mapping a task with a robustness lower than the threshold required to prevent dropping means that unless a task is dropped ahead of that mapped task, it will be dropped during the next engagement of the task dropper. As shown in the section on Pruning Threshold Awareness in Chapter 4, this leads to poor performance. This leads directly to the idea that requiring a *higher* threshold to map tasks than to drop tasks will be beneficial to the robustness of the system, as the Pruner will wait for more robust mapping pairs, and will have some level of slack between the requirements to map a task, and the

threshold required to prevent dropping a task.

Figure 6.1. Percentage of tasks meeting their deadlines (vertical axis) examining the impact of different deferring and dropping thresholds. Dropping toggle is set to 1 missed task. Horizontal axis shows the deferring threshold. Dropping threshold is denoted by line color



A small constant is added to the dropping threshold when considering whether to defer a task (*e.g.,* a dropping threshold of 50% would require 55% robustness to map a task to a machine). To test this, three dropping thresholds (25%, 50%, and 75%) are examined in an experiment increasing the modifier on each by 5% until the deferring threshold becomes 90%. This is done at an extreme level of oversubscription.

6.2.1 Performance evaluation of decoupled thresholds. Figure 6.1 shows that using a higher deferring threshold than dropping threshold leads to better system robustness, up to a point. For the case of a 25% dropping threshold, there is an inflection point after which increasing the deferring threshold leads to a decrease in robustness. The results are difficult to judge between differnt dropping thresholds, as the difference in results lack statistical significance at 30 trials. The inflection point of the lower dropping threshold suggests that while decoupling the two thresholds can be beneficial, too much decoupling with a low dropping threshold can cause negative consequences, as this can allow tasks to continue using resources past the point when they would best be dropped. The data points to using a 50% or 75% dropping threshold and a 75%-90% deferring threshold under this level of extreme oversubscription, but more trials need to be run in order to see significant difference within the confidence of the mean.

6.3 Dynamic Per-Task Dropping Threshold

When it is time to consider whether a task should be probabilistically dropped, the Pruner discussed in Chapter 4 treats all tasks the same. The probability of completion on time is calculated, checked against the pruning threshold, and then dropped or not, accordingly. However, not all tasks have the same effects on the probability of on-time completion for the tasks behind them in queue. This can be taken into account to make the best decision about which tasks get to stay and which are dropped. When convolving the PMFs of tasks, aside from the probability of on-time completion, the two task-level characteristics that impact the effect of a given task on

the probability of those tasks behind it are the position of the task in queue, and the shape of its PMF.

The closer a task is to execution, the more tasks are affected by its completion time. With a queue size of six, an executing task affects the completion time of five tasks, where the execution time of a task at the end of the queue affects no tasks. The closer a task is to execution, the more tasks affected by the completion time of a task, the higher the robustness threshold for dropping.

Using the skewness and queue position, the system can adjust the probability threshold for dropping a task dynamically, for each task. Ideally this will allow more tasks to complete before their deadline, leading to a higher HC system robustness.

$$\frac{-s \times \beta}{j+1}$$

Calculating the normalized skewness of a task is performed by the formula:

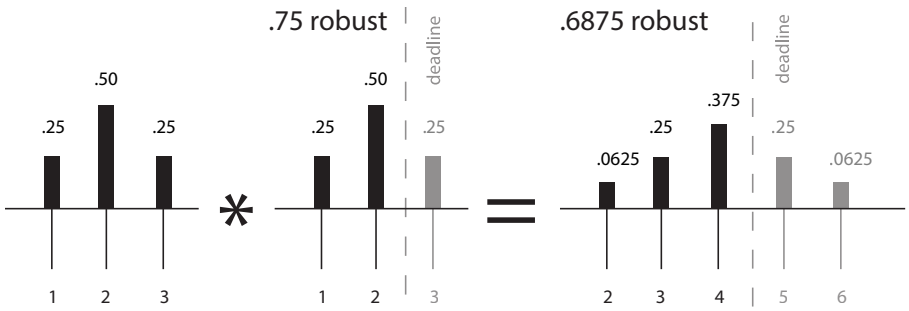
$$S = \frac{\sqrt{N(N-1)}}{N-2} \times \frac{\sum_{i=1}^{n} (Y_i - \bar{Y})^3 / N}{\sigma^3}$$

This dynamic adjustment of the required probability is done only in the dropping stage of the Pruner. When it comes to deferring tasks, the position of the task is always the same (*i.e.,* the tail of the queue), and it is too early to consider the shape of the tasks PMF, as there are, as yet, no tasks behind it in queue.

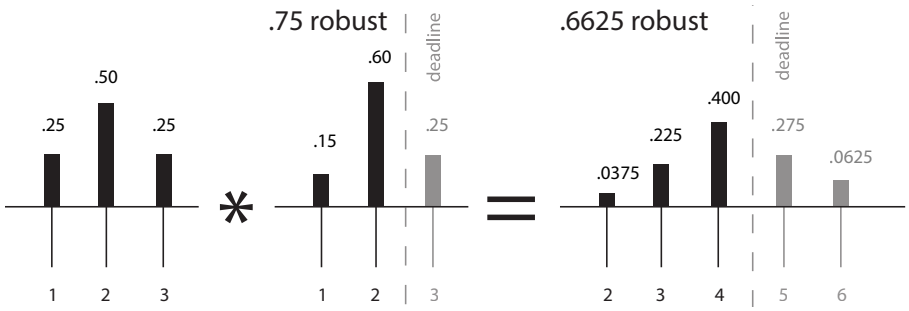
6.3.1 Performance evaluation of per-task dynamic pruning threshold.

Figure 6.3 shows that with this task arrival list, in this system, for these trials, the per-time thresholds result in no statistical difference in robustness. The machines and tasks in this thesis come from benchmarks, which coupled with the arrival rates

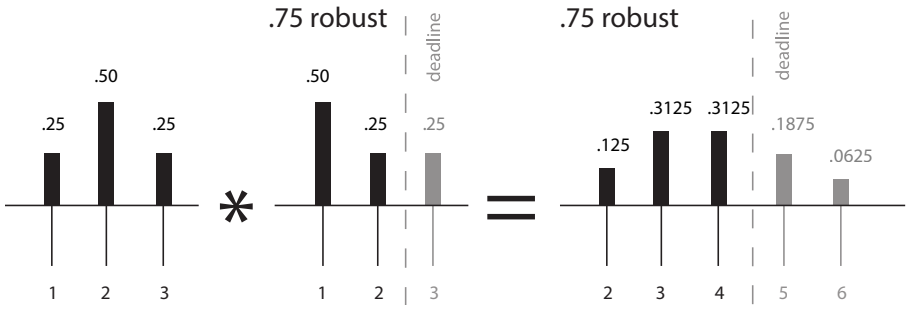
Figure 6.2. A demonstration of three types of skews' effects on CTD convolutions



(a) No Skew

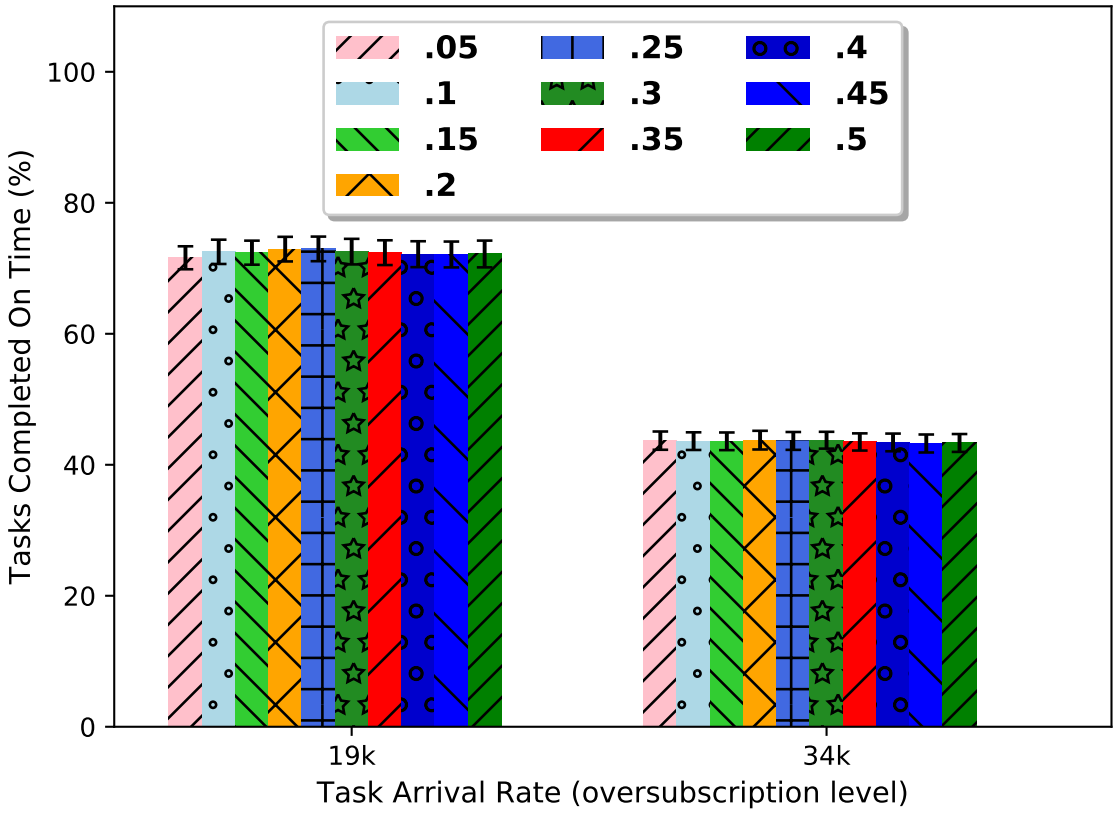


(b) Left Skew



(c) Right Skew

Figure 6.3. Percentage of tasks meeting their deadlines (vertical axis) examining the impact of a per-task dropping threshold. Each bar shows a different β value (the weight of the dynamic threshold). Dropping toggle is set to 1 missed task. Pruning threshold is set to 75%. The horizontal axis is the level of oversubscription in tasks per period.



and patterns used, result in PMFs for task-machine mappings with performance characteristics that hide the potential effectiveness of such a system. To more deeply analyze the potential benefits of a per-task dropping threshold, further research needs to be done using carefully crafted tasks/machines/arrival patterns.

6.4 Interpreting the Dropping Toggle

Under certain circumstances, it is conceivable that a system reaching the dropping-toggle value only signals an acute spike in task-arrival, and not an ongoing

state of oversubscription. To test this, a Pruner is developed that does not react to temporary spikes in deadline misses, but only to sustained oversubscription. By using a weighted average of deadline misses, the Pruner will be slower to engage, and slower to disengage.

To judge the oversubscription state in the system, the Pruner operates based on moving weighted average number of tasks that missed their deadlines during the past mapping events. Let d_τ denote the oversubscription level of the HC system at mapping event τ; and m_τ denote the number of tasks missing their deadline since the past mapping event. Parameter λ is tunable and is determined based on the relative weight assigned to the past events.

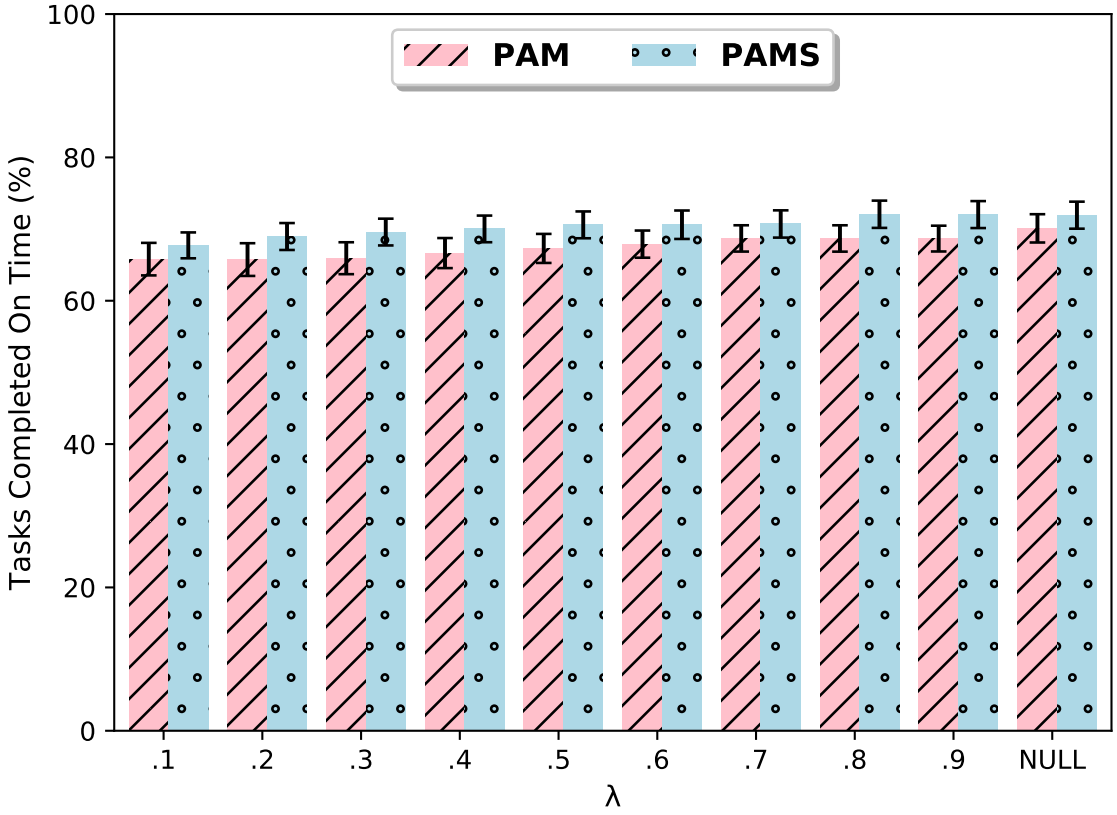
$$d_\tau = m_\tau \times \lambda + d_{\tau-1} \times (1 - \lambda)$$

Another potential concern is minor fluctuations about the toggle switching the dropping off and then back on. To prevent minor fluctuations around a threshold to affect the state of a switch or toggle, circuit designers use employ a Schmitt Trigger to smooth things out. The mechanical details of such a physical device are outside the scope of this thesis, but the concept of requiring different values as a trigger based on trigger state is tested in this section.

The default toggle is tested against a schmitt-style toggle, and the results are compared at different arrival rates.

6.4.1 Performance evaluation of different dropping toggle interpretations. Figure 6.4 shows that, for the effect of λ on robustness, as more weight is given to the history of tasks missing their deadline, the number of tasks in the

Figure 6.4. Percentage of tasks meeting their deadlines (vertical axis) examining the impact of toggle-history and minor fluctuations about the toggle. Dropping toggle is set to 1 missed task. Pruning threshold is set to 75%. Horizontal axis shows the λ value (weight of current vs. historical data). λ of NULL means no history is used.



system completing on time is decreased. The highest percentage of successful tasks occurs when the least weight is given to the history of deadline misses, when the current number of missed tasks, alone, is used to decide whether to engage the dropping mechanism. This is due in part to the steady nature of task-arrival in the system, such that when one task fails to meet its deadline, there is a high chance that more unsuitable tasks remain in the machine queues and executing on the machines, necessitating dropping.

The mean of the results also show that having a lower threshold to turn dropping off, once engaged, also results in more tasks successfully completing overall for each value of λ used, though the difference at some λ values is of no statistical significance at this number of trials. This shows that under high levels of oversubscription, the best results come from taking immediate action when tasks miss their deadline, and then a steady application of dropping tasks probabilistically until the situation is decidedly controlled (*i.e.,* until the lower bound of the the schmitt-trigger is reached).

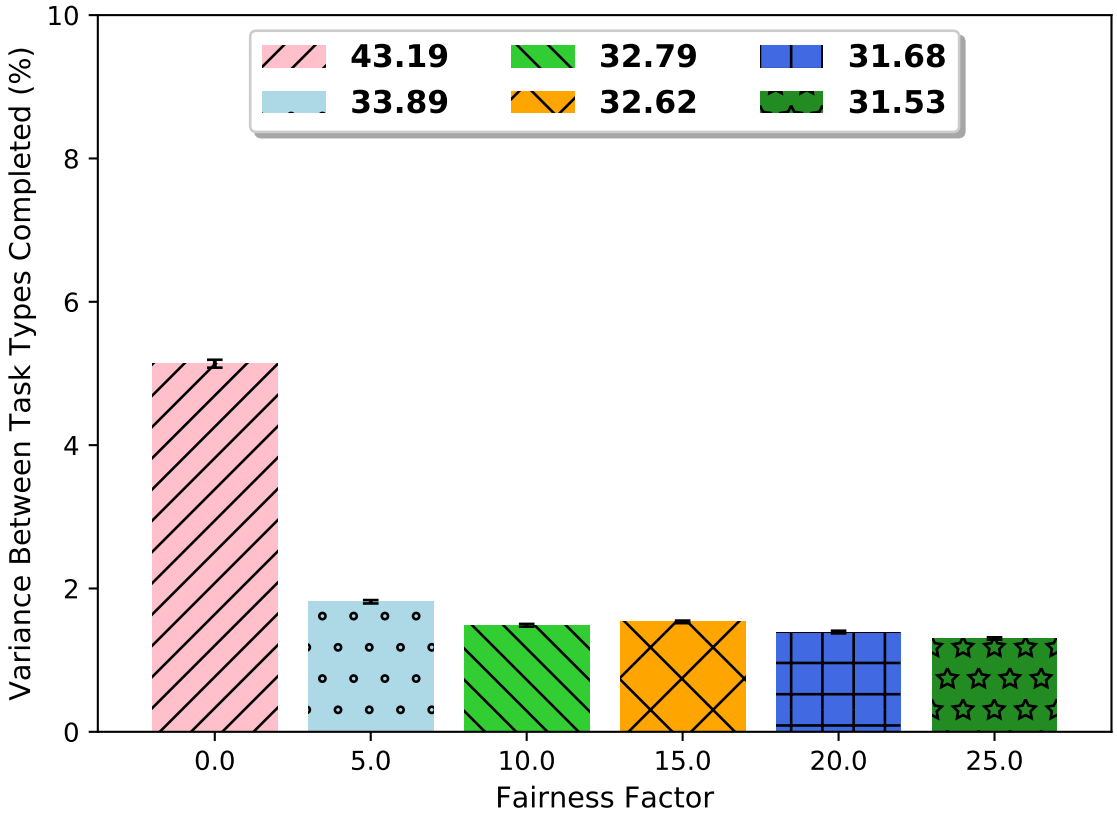
6.5 Evaluating Fairness among Task Types

When probabilistically dropping tasks, it is possible, and in some situations likely, for the system to favor smaller tasks, as those shorter tasks usually have a higher probability of completion within their deadline. In some systems (such as live video streaming) it is reasonable that dropping all long tasks of a certain type would be detrimental to the system (such as if all resizing happened, but conversion between encoding formats was deemed to risky to be executed).

As an initial investigation into engendering fairness amongst the types of tasks completed by the system, a method to track the dropping of task types, and to adjust the required probability for tasks is described. A modifying value is tracked for each task type in a system. Each time a task is dropped, the corresponding value is increased, and each time a task completes, its corresponding value is decreased. By subtracting this value from the threshold for pruning (both in deferring and dropping), the pruning mechanism attempts to create a more fair distribution of completed tasks by allowing certain tasks to be mapped or to be protected from dropping.

For this experiment, the percentage of each task type completing on time is tracked. The objective is to minimize the variance among these. The overall robustness of the system is also tracked for each, to understand the cost in robustness that fairness requires in these circumstances.

Figure 6.5. Percentage of tasks meeting their deadlines (vertical axis) examining the impact of different levels of task-fairness adjustment. Dropping toggle is set to 1 missed task. Pruning threshold is set to 75%. Horizontal axis shows percentage added to the tracked per-task pruning threshold when tasks of that type fail to complete on time. Each bar is labeled with the system robustness under that fairness implementation.



6.5.1 Performance evaluation of fairness technique. Figure 6.5 shows that while some tiny measure toward fairness can be attained, the result is fewer tasks completed overall. Using the most conservative system implemented (a 5% modifier)

resulted in a 25% reduction in overall system robustness (from 40% to 30%), while reducing variance in completed tasks from 5% to 2%. This resultant reduction in robustness is because, as deferring tasks is the most significant contributor to the success of the system, deferring fewer tasks in an attempt at fairness results in fewer tasks successfully reaching their deadline overall. Further increasing the amount of the fairness modifier results in a significant increase in fairness while resulting in much smaller reductions in overall robustness.

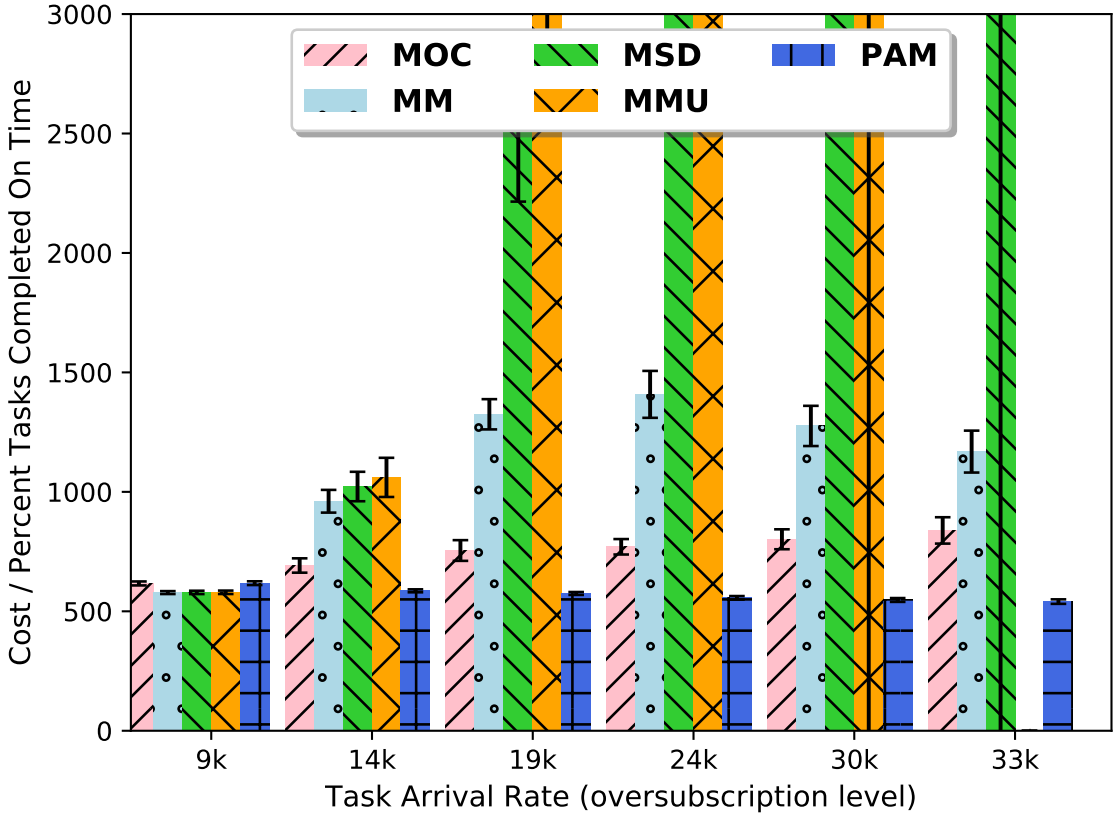
6.6 Costs Comparison of PAM and Other Mapping Heuristics

The focus of this thesis is maximizing on-time completions of tasks in a heterogeneous distributed computing system, however there are other metrics of success to consider; one of these is cost. Time spent computing tasks that fail to successfully complete is a waste of computing resources that, in addition to leading to compounding deadline misses, leads to wasted use of resources which have costs associated (*e.g.,* electricity and cooling). For certain scenarios (*i.e.,* cloud computing) this can be counted in dollars (or your currency of choice).

To investigate this, pricing from Amazon Web Services VMs has been mapped to the machines in the simulation. Each machine's usage time is tracked. The price incurred to process the tasks is divided by the percentage of on-time tasks completed to give a normalized view of other costs of the system.

6.6.1 Cost evaluation of probabilistic dropping. Figure 6.6 shows that PAM incurs a smaller cost per completed task than other heuristics when in an oversubscribed state. When the arrival rate is at 9k, PAM performs a little worse, as in

Figure 6.6. Percentage of tasks meeting their deadlines (vertical axis) examining the impact of probabilistic pruning on costs in an HC cloud system at differing levels of oversubscription. Dropping toggle is set to 1 missed task. Pruning threshold is set to 75%. The horizontal axis is the level of oversubscription in tasks per period.



an effort to ensure the highest success rate, the best machines are always chosen (*i.e.*, often the most expensive machines), resulting in the same number of tasks completed for a higher cost. However, as the arrival rate increases, the relative cost of PAM decreases. At extreme levels of oversubscription, the difference between heuristics such as MMU and MSD and PAM become unchartable, as MSD and MMU both prioritize tasks least likely to succeed, whereas PAM prioritizes those most likely to succeed. In the highest tested levels of oversubscription, MMU performed so poorly that zero tasks

completed in any trial, resulting in a cost of zero. While previous tests have shown PAM to be outperforming other heuristics in terms of robustness in the face of oversubscription, these results show that in most levels of oversubscription, the benefits are realized in dollar cost as well, due to not processing tasks needlessly.

6.7 Summary

In this chapter, decoupling the deferring and dropping threshold is shown to have a significant positive effect on the number of successfully completed tasks. In this study, toggling task dropping on and off works best when done decisively. While moving toward fairness amongst task types, in regards to dropping tasks, is possible, doing so comes at the drastic reduction in tasks completing on time overall. Results show that while there is some slight variation in system robustness with the dynamic per-task threshold, it is not statistically significant. Finally, the cost-benefit of using the Pruning Aware Mapper become more pronounced as the level of oversubscription increases. The next chapter, the last, will offer conclusions and future work.

Chapter 7: Conclusion and Future Works

This chapter summarizes the research and findings of this thesis. Additionally, those further research topics that emerged during the course of this research but were not discussed in this thesis are discussed. The goal of this research was to evaluate the impact of pruning tasks with low probability of success on the robustness of an HC system.

7.1 Discussion

In chapter 4, a probability was determined that can be used by mapping heuristics in oversubscribed HC systems to either map or defer a task. It was also determined how pruned tasks should be treated based on oversubscription level of the system. Specifically, it was shown that tasks with low chance of success should be deferred (be given another chance for more favorable mapping in the next scheduling event). Alternatively, when the system is sufficiently oversubscribed, the tasks with low chance of success must be dropped to alleviate the oversubscription and increase the probability of other tasks succeed. Finally, a mapping method was developed that operates based on the probabilistic pruning of the tasks. Evaluation results revealed that PAM, the pruning-aware mapper was successful in increasing the number of tasks meeting their deadline in a simulated oversubscribed system with probabilistic task dropping. Even with no probabilistic dropping in the system, the ability to defer tasks during the mapping event led to a 40% increase in robustness compared to the baseline heuristics. Results also show that, probabilistically dropping tasks can allow an oversubscribed system to recover from otherwise-poor mapping events made by

heuristics with no awareness of the pruning mechanism, and increases the number of tasks that meet their deadline in a system. This success becomes especially pronounced in more extreme levels of oversubscription.

In chapter 5, probabilistic deferring and dropping mechanisms from chapter 4 were applied to mapping other dynamic mapping heuristics. It was shown that both forms of pruning are beneficial as modules that can be added to an existing system. Evaluation results show that immediate-mode heuristics can benefit from probabilistic dropping, as it allows them the ability to recover from quick mapping decisions that turn out to not be very robust. It was also shown that when using a single pass, immediate mode, heuristic based on robustness (MaxRobust), adding a pruner to the machine queues can enable it to perform on par with batch mode heuristics such as MinMin, even under extreme levels of oversubscription.

In chapter 6, further refinements were made to the pruning mechanism: the deferring and dropping thresholds were decoupled. This allowed tasks to be deferred requiring a higher probability to schedule, and a lower to drop, resulting in more tasks completing on-time in the system. Additionally, a method for dynamically adjusting the robustness threshold for dropping tasks based on per-task characteristics was designed and tested(*e.g.,* place in machine queue, and shape of CTD). Initial results did not result in any statistically significant increase in the successful completion of tasks, and further work must be done. In an attempt to prevent certain task types to be starved from the system, a fairness-enabling mechanism was proposed and investigated. By allowing task types that were dropped from the system to have leniency when it

comes to dropping and deferring tasks did result in a more uniform distribution of task types completing. Though effective in reducing the variance among the types of tasks completing on time, that fairness came at the cost of greatly reduced tasks completing on time in the system as a whole. A schmitt-trigger like toggle was created to smooth out the engagement of the task-dropping mechanism, as well as using a moving average as a representation of the level of oversubscription. The schmitt-trigger approach to toggling led to an increase in system robustness, however incorporating the history of tasks meeting their deadline in any fashion resulted in decreased system robustness. Finally, the cost-benefits of using the Pruning Aware Mapping system were examined by comparing the costs / percent of tasks completed on time against other benchmark heuristics. While pruning tasks can result in higher costs when undersubscribed, the cost benefits become apparent and grow as the oversubscription level rises.

7.2 Future Works

7.2.1 Dynamic per-task dropping threshold via PET and workload synthesis.

Though the results of dynamic per-task dropping thresholds did not show statistically significant differences from a system-wide threshold, there is much room for exploration in this area. Carefully crafting a variety of PET matrix PMFs to explore different types of inconsistently heterogeneous systems is the first step. Another factor to consider is the pattern of task arrival. An HC system whose tasks arrive in a low-varying rate will respond to dropped tasks differently than an HC system whose tasks arrive in discrete bursts. Understanding the effects of a dynamic per-task probabilistic task dropping threshold will require examination of these and other

characteristics of the system.

7.2.2 Tasks with priority. In this thesis, all tasks are droppable, and each task is exactly as important as any other task in the system. In many HC systems, this is not the case. A mechanism to account for and weigh this priority will need to be developed. A future study will be necessary to investigate the effect of probabilistic task dropping on HC systems which implement a task priority system, in which the completion of some tasks are more valuable than others, and some tasks must complete (cannot be dropped).

7.2.3 Preemption of running tasks. This work is concerned with tasks that can be dropped in an attempt to maximize system robustness. A future plan is to extend the probabilistic approach to consider the saving of task-state and preemption of tasks. This would start with an examination of preemption's impact on the convolution process, and require an estimation of time-to-preempt to simulate preemptable tasks, and the effect on tasks meeting their deadlines.

7.2.4 Approximate computation. Not all tasks require full and accurate completion. A future avenue of research is to add a method to approximately compute those tasks that are unlikely to meet their deadlines, instead of dropping them from the system. This will require the development of an additional feature of the pruning mechanism that has an awareness of approximate computation options, to select the appropriate computation method, in an inconsistently heterogeneous system.

7.2.5 Defer pool for immediate mode heuristics. Because of the increase in the performance added by probabilistically dropping tasks when using immediate

mode heuristics, and the positive impact of deferring task-machine mapping until high-affinity machines become available, a way to allow immediate mode heuristics to be deferred should be investigated. Some sort of pool or deferred queue can be added to the immediate mode heuristics to test the effect of probabilistically deferring using the simpler heuristics.

7.2.6 Optimizing implementation and analyzing overhead of pruning mechanism. Convolving probability mass functions is not a trivial operation. Capturing this time-cost and integrating it into the calculations will help to better understand how this mechanism can be applied to maximize system robustness. Optimizing this procedure will also make running experiments easier, as the process is time-consuming, and implementation optimizations have been outside of the scope of this thesis.

7.2.7 Mitigating convolutional complexity with growing machine queue sizes. In this work, the machine-queue sizes for batch mode heuristics was kept to a relatively low size of 6, including the executing task. This was done to reduce the compound uncertainty resulting from long queue times, but this can result in tasks remaining in the batch queue for a long time, potentially creating a bottleneck in constantly re-calculating deferring probabilities. A way to approximate the calculations, or perhaps memoize their results until changes in state require recalculation, or otherwise deal with the computational time will be necessary to increase the queue sizes past a certain point.

Bibliography

[1] 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*, vol. Special Tamkang University 50th Anniversary Issue, 3, no. 3, pp. 195–208, Nov. 2000, invited.

[2] M. Zahran, "Heterogeneous computing: Here to stay," *Queue*, vol. 14, no. 6, p. 40, 2016.

[3] H.-E. Zahaf, A. E. H. Benyamina, R. Olejnik, and G. Lipari, "Energy-efficient scheduling for moldable real-time tasks on heterogeneous computing platforms," *Journal of Systems Architecture*, vol. 74, pp. 46–60, 2017.

[4] Q. Zhao, M. Amagasaki, M. Iida, M. Kuga, and T. Sueyoshi, "A study of heterogeneous computing design method based on virtualization technology," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 4, pp. 86–91, 2017.

[5] Amazon, "Amazon Web Sevices (AWS) Instance Types," 2018. [Online]. Available: https://aws.amazon.com/ec2/instance-types/

[6] 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 (JPDC)*, vol. 59, no. 2, pp. 107–131, Nov. 1999.

[7] T. D. Braun, H. J. Siegel, N. Beck, L. Bölöni, 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 (JPDC)*, vol. 61, no. 6, pp. 810–837, June 2001.

[8] X. Li, M. A. Salehi, M. Bayoumi, N.-F. Tzeng, and R. Buyya, "Cost-efficient and robust on-demand video stream transcoding using heterogeneous cloud services," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 29, no. 3, pp. 556–571, Mar. 2018.

[9] Z. Zong, R. Ge, and Q. Gu, "Marcher: A heterogeneous system supporting energy-aware high performance computing and big data analytics," *Big Data Research*, vol. 8, pp. 27–38, 2017.

[10] Louisiana Optical Network Infrastructure, "LONI Resources QB2," 2018. [Online]. Available: http://hpc.loni.org/resources/hpc/system.php?system=QB2

[11] NextPlatform, "Heterogeneous Supercomputing on Japan's Most Powerful System," 2018. [Online]. Available: https://www.nextplatform.com/2017/08/28/heterogeneous-supercomputing-japans-powerful-system/

[12] J. Smith, V. Shestak, H. J. Siegel, S. Price, L. Teklits, and P. Sugavanam, "Robust resource allocation in a cluster based imaging system," *Parallel Computing*, vol. 35, no. 7, pp. 389–400, July 2009.

[13] V. Shestak, J. Smith, A. A. Maciejewski, and H. J. Siegel, "Stochastic robustness metric and its use for static resource allocations," *Journal of Parallel and Distributed Computing (JPDC)*, vol. 68, no. 8, pp. 1157–1173, Aug. 2008.

[14] M. A. Salehi, J. Smith, A. A. Maciejewski, H. J. Siegel, E. K. Chong, J. Apodaca, L. D. Briceño, T. Renner, V. Shestak, J. Ladd *et al.*, "Stochastic-based robust dynamic resource allocation for independent tasks in a heterogeneous computing system," *Journal of Parallel and Distributed Computing (JPDC)*, vol. 97, pp. 96–111, Nov. 2016.

[15] X. Li, M. A. Salehi, and M. Bayoumi, "VLSC: Video Live Streaming Using Cloud Services," in *Proceedings of the 6th IEEE International Conference on Big Data and Cloud Computing Conference*, ser. BDCloud '16, October 2016, pp. 595–600.

[16] ——, "High performance on-demand video transcoding using cloud services," in *Proceedings of the 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, ser. CCGrid '16. IEEE, 2016, pp. 600–603.

[17] P. Liu, X. Li, J. J. Qu, W. Wang, C. Zhao, and W. Pichel, "Oil spill detection with fully polarimetric uavsar data," *Marine Pollution Bulletin*, vol. 62, no. 12, pp. 2611–2618, 2011.

[18] L. Szalinski, L. Abdulkareem, M. Da Silva, S. Thiele, M. Beyer, D. Lucas, V. H. Perez, U. Hampel, and B. Azzopardi, "Comparative study of gas–oil and gas–water two-phase flow in a vertical pipe," *Chemical engineering science*, vol. 65, no. 12, pp. 3836–3848, 2010.

[19] M. Fingas and C. E. Brown, "Oil spill remote sensing: a review," in *Oil spill science and technology*. Elsevier, 2011, pp. 111–169.

[20] W. Ma, L. Cao, L. Yu, G. Long, and Y. Li, "Gpu-fv: Realtime fisher vector and its applications in video monitoring," in *Proceedings of the 2016 ACM on International Conference on Multimedia Retrieval*. ACM, 2016, pp. 39–46.

[21] U. Y. Ogras and R. Marculescu, "Analysis and optimization of prediction-based flow control in networks-on-chip," in *Modeling, Analysis and Optimization of Network-on-Chip Communication Architectures*. Springer, 2013, pp. 105–133.

[22] M. Airouche, L. Bentabet, and M. Zelmat, "Image segmentation using active contour model and level set method applied to detect oil spills," in *Proceedings of the World Congress on Engineering*, vol. 1, no. 1. Lecture Notes in Engineering and Computer Science, 2009, pp. 1–3.

[23] B. Khemka, R. Friese, L. D. Briceo, H. J. Siegel, A. A. Maciejewski, G. A. Koenig, C. Groer, G. Okonski, M. M. Hilton, R. Rambharos, and S. Poole, "Utility functions and resource management in an oversubscribed heterogeneous computing environment," *IEEE Transactions on Computers*, vol. 64, no. 8, pp. 2394–2407, Aug 2015.

[24] B. Khemka, R. Friese, S. Pasricha, A. A. Maciejewski, H. J. Siegel, G. A. Koenig, S. Powers, M. Hilton, R. Rambharos, and S. Poole, "Utility driven dynamic resource management in an oversubscribed energy-constrained heterogeneous system," in *Proceedings of the 28th IEEE International Parallel & Distributed Processing Symposium Workshops*, ser. IPDPSW '14, May 2014, pp. 58–67.

[25] ——, "Utility maximizing dynamic resource management in an oversubscribed energy-constrained heterogeneous computing system," *Sustainable Computing: Informatics and Systems*, vol. 5, pp. 14–30, Mar. 2015.

[26] S. AlEbrahim and I. Ahmad, "Task scheduling for heterogeneous computing systems," *The Journal of Supercomputing*, vol. 73, no. 6, pp. 2313–2338, 2017.

[27] E. Coffman and J. Bruno, *Computer and Job-shop Scheduling Theory*. New York, NY: John Wiley & Sons, 1976.

[28] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on non-identical processors," *Journal of the ACM*, vol. 24, no. 2, pp. 280–289, Apr. 1977.

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

[30] K. Li, X. Tang, and K. Li, "Energy-efficient stochastic task scheduling on heterogeneous computing systems," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 25, no. 11, pp. 2867–2876, Nov. 2014.

[31] Y. Chen, H. C. Liao, and T. Tsai, "Online real-time task scheduling in heterogeneous multicore System-on-a-Chip," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 24, no. 1, pp. 118–130, Jan. 2013.

[32] Y. Lee and A. Y. Zomaya, "A novel state transition method for metaheuristic-based scheduling in heterogeneous computing systems," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 19, no. 9, pp. 1215–1223, Sep. 2008.

[33] Y. C. Lee and A. Y. Zomaya, "Rescheduling for reliable job completion with the support of clouds," *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1192–1199, Oct. 2010.

[34] C. Delimitrou and C. Kozyrakis, "QoS-aware scheduling in heterogeneous datacenters with Paragon," *ACM Transactions on Computer Systems*, vol. 31, no. 4, pp. 1–34, Dec. 2013.

[35] ——, "Quality-of-Service-aware scheduling in heterogeneous data centers with paragon," *IEEE Micro*, vol. 34, no. 3, pp. 17–30, 2014.

[36] A. Tumanov, T. Zhu, J. W. Park, M. A. Kozuch, M. Harchol-Balter, and G. R. Ganger, "Tetrisched: global rescheduling with adaptive plan-ahead in dynamic heterogeneous clusters," in *Proceedings of the Eleventh European Conference on Computer Systems*, ser. EuroSys '16. ACM, 2016, p. 35.

[37] D. Machovec, S. Pasricha, A. A. Maciejewski, H. J. Siegel, G. A. Koenig, M. Wright, M. Hilton, R. Rambharos, T. Naughton, and N. Imam, "Preemptive resource management for dynamically arriving tasks in an oversubscribed heterogeneous computing system," in *Proceedings of the 31st IEEE International Parallel and Distributed Processing Symposium Workshops*, ser. IPDPSW '17, May 2017, pp. 54–64.

[38] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Algorithms for cost-and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds," *Future Generation Computer Systems*, vol. 48, pp. 1–18, 2015.

[39] K. Li, "Performance analysis of list scheduling in heterogeneous computing systems," *World Academy of Science, Engineering and Technology, International Journal of Computer, Electrical, Automation, Control and Information Engineering*, vol. 4, no. 3, pp. 484–491, 2010.

[40] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling," in *Proceedings of the 5th European conference on Computer systems*. ACM, 2010, pp. 265–278.

[41] T. W. Malone, R. E. Fikes, and M. T. Howard, "Enterprise: A market-like task scheduler for distributed computing environments," 1983.

[42] L. Wasserman, *All of Statistics: A Concise Course in Statistical Inference*. New York, NY: Springer Science+Business Media, 2005.

[43] B. D. Young, J. Apodaca, L. D. Briceño, J. Smith, S. Pasricha, A. A. Maciejewski, H. J. Siegel, B. Khemka, S. Bahirat, A. Ramirez, and Y. Zou, "Deadline and energy constrained dynamic resource allocation in a heterogeneous computing environment," *The Journal of Supercomputing*, vol. 63, no. 2, pp. 326–347, Feb. 2013.

[44] A. Dogan and F. Ozguner, "Genetic algorithm based scheduling of meta-tasks with stochastic execution times in heterogeneous computing systems," *Journal of Cluster Computing*, vol. 7, no. 2, pp. 177–190, Apr. 2004.

[45] J. Cao, K. Li, and I. Stojmenovic, "Optimal power allocation and load distribution for multiple heterogeneous multicore server processors across clouds and data centers," *IEEE Transactions on Computers*, vol. 63, no. 1, pp. 45–58, Jan. 2014.

[46] A. Kumar and R. Shorey, "Performance analysis and scheduling of stochastic fork-join jobs in a multicomputer system," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 4, no. 10, pp. 1147–1164, Oct. 1993.

[47] X. He, X. Sun, and G. Von Laszewski, "Qos guided min-min heuristic for grid task scheduling," *Journal of Computer Science and Technology*, vol. 18, no. 4, pp. 442–451, 2003.

[48] M. Pedemonte, P. Ezzatti, and Á. Martín, "Accelerating the min-min heuristic," in *Parallel Processing and Applied Mathematics*. Springer, 2016, pp. 101–110.

[49] P. Ezzatti, M. Pedemonte, and Á. Martín, "An efficient implementation of the min-min heuristic," *Computers & Operations Research*, vol. 40, no. 11, pp. 2670–2676, 2013.

James A. S. Gentry Bachelor of Arts, University of Massachusetts, Spring 2005; Master of Business Administration, University of Louisiana at Lafayette, Fall 2007; Master of Science, University of Louisiana at Lafayette, Summer 2018
Major: Computer Science
Title of Thesis: Robust Resource Allocation of Independent Tasks in Heterogeneous Computing Systems via Probabilistic Task Pruning
Thesis Director: Mohsen Amini Salehi
Pages in Thesis: 88; Words in Abstract: 290

Abstract

In heterogeneous distributed computing system, diversity can be present both in the computational resources and in the types of arriving tasks. In an inconsistently Heterogeneous Computing (HC) system, different task types can have different performance characteristics (*i.e.,* execution times) on heterogeneous machines. A mapping method is required to match arriving tasks with machines based on both machine availability and performance to maximize the number of tasks meeting their deadlines (known as robustness). In particular, for tasks with hard individual deadlines (*e.g.,* live video streaming tasks), those that have missed their deadlines are dropped, as there is no value in executing them. The problem investigated in this research is how to maximize robustness of an HC system, specifically, when it is oversubscribed. The proposal is to prune (*i.e.,* defer or drop) tasks with low probability of meeting their deadlines. Pruning low-chance tasks increases the probability of other tasks meeting their deadlines. To that end, first a model is provided to estimate the probability of meeting deadline for each task in the presence of task dropping. Second, a pruning mechanism is proposed to predictively defer or drop tasks in an effort to maximize the overall robustness of the HC system. Third, a mapping method is proposed that

functions based on the pruning mechanism and improves robustness of the HC system. Fourth, to show a broad application, the pruning mechanism is applied to other mapping heuristics. Fifth, further development of the pruning mechanism is made from multiple fronts to improve robustness, engender fairness amongst completed task types, and examine the cost ramifications of using a pruning mechanism. Simulation results, harnessing a selection of mapping heuristics, show the efficacy of the proposed pruning mechanism can improve robustness of some oversubscribed HC system by more than 40%.

Biographical Sketch

James A. S. Gentry received his Bachelor of Arts in the spring of 2005 in English from the University of Massachusetts Boston, and his M.B.A. from the University of Louisiana at Lafayette in 2007. The son of Robert and Judith Gentry, he began his pursuit of a master's degree in the fall of 2017 at the University of Louisiana at Lafayette and received a Graduate Teaching Assistantship–first helping students learn the basics of computer science, and then helping them through the crucible of operating systems. He completed the requirements for the degree in summer of 2018.