Efficient Task Pruning Mechanism to Improve Robustness of Heterogeneous Computing Systems

Chavit Denninnart[1,*], James Gentry[1,**], Ali Mokhtari[1,*], Mohsen Amini Salehi[1,*]

Abstract

In heterogeneous distributed computing (HC) systems, diversity can exist in both computational resources and arriving tasks. In an inconsistently heterogeneous computing system, task types have different execution times on heterogeneous machines. A method is required to map arriving tasks to machines based on machine availability and performance, maximizing the number of tasks meeting deadlines (defined as *robustness*). For tasks with hard deadlines (*e.g.,* those in live video streaming), tasks that miss their deadlines are dropped. The problem investigated in this research is maximizing the robustness of an oversubscribed HC system. A way to maximize this robustness is to prune (*i.e.,* defer or drop) tasks with low probability of meeting their deadlines to increase the probability of other tasks meeting their deadlines. In this paper, we first provide a mathematical model to estimate a task's probability of meeting its deadline in the presence of task dropping. We then investigate methods for engaging probabilistic dropping. We propose methods to dynamically determine task dropping and deferring threshold probabilities. Next, we develop a pruning system and a pruning-aware mapping heuristic, which we extend to engender fairness across various task types. We present the pruning mechanism as an independent component that can be applied to any mapping heuristic to improve the system robustness. To reduce overhead of the pruning mechanism, we propose approximation methods that remarkably reduce the number of mathematical calculations and improve the practicality of deploying the mechanism in heterogeneous or even homogeneous computing systems. We show the cost and energy gains of the pruning mechanism. Simulation results, harnessing a selection of mapping heuristics, show efficacy of the pruning mechanism in improving robustness (on average by $\simeq$22%) and cost in an oversubscribed HC system by up to $\simeq$33%.

Keywords:
Heterogeneous Computing (HC), Probabilistic Pruning, Mapping Heuristic, Robustness

[*]School of Computing and Informatics, University of Louisiana at Lafayette, Louisiana, USA
[**]Spectra Logic, Colorado, USA
Email addresses: chavit.denninnart1@louisiana.edu (Chavit Denninnart), gentry@hpcclab.org (James Gentry), ali.mokhtari1@louisiana.edu (Ali Mokhtari), mohsen.aminisalehi@louisiana.edu (Mohsen Amini Salehi)

1. Introduction

A Heterogeneous Computing (HC) system can be described by two types of heterogeneity: inconsistent and consistent[1,2]. Inconsistent machine heterogeneity refers to differences in machine architecture (*e.g.,* CPU versus GPU versus FPGA[3,4,5]). Consistent machine heterogeneity describes the differences among machines of a certain architecture (*e.g.,* different clock speeds). Compute services offered by cloud providers are a good example of an HC system. Amazon cloud[6] offers inconsistent heterogeneity in form of various Virtual Machine (VM) types, such as CPU-Optimized, Memory-Optimized, Disk-Optimized, and Accelerated Computing (GPU and FPGA). Within each type, various VMs are offered with consistent performance scaling with price[6]. Moreover, both consistent and inconsistent heterogeneity can exist in arriving tasks. For example, an HC system dedicated to processing live video streams is responsible for many categorically different types of tasks: changing video stream resolution, changing the compression standard, changing video bit-rate[2]. Each of these task types can be consistently heterogeneous within itself (*e.g.,* it takes longer to change resolution of 10 seconds of video, compared to 5).

Many HC systems (*e.g.,* [7,8]) present both consistent and inconsistent heterogeneity in machines used and task types processed[9]. These systems present cases where each task type can execute differently on each machine type, where machine type *A* performs task type 1 faster than machine type *B* does, but is slower than other machine types for task type 2. Specifically, compute intensive tasks run faster on (*i.e.,* matches better with) a GPU machine whereas tasks with memory and disk accesses bottlenecks (*e.g.,* in-memory databases[10,11,12]) runs faster on a CPU-based machine.

All of this heterogeneity results in uncertainty for a given task's execution time, thus, inefficiency of resource allocation[1]. Accordingly, a major challenge in HC systems is to assign tasks to machines to optimize performance goal of the system[1]. We define *robustness* as the degree to which a system can maintain performance in the face of uncertainty[13]. The overall *goal* of this study is to maximize the robustness of an HC system.

Each task is considered to have a hard individual deadline, past which, no value remains in executing the task. Hence, tasks are dropped (*i.e.,* removed) from the system when their deadline passes[14,15]. When the HC system is under load, such that it is impossible for all tasks to complete before their deadlines, the system is considered *oversubscribed*. The performance metric based on which we measure robustness of an HC system is the number of tasks that meet their deadlines in the system. Therefore, the specific goal of this study is to maximize the number of tasks meeting their deadlines in the HC system (referred to as *task success*) in the face of uncertain execution times in an oversubscribed system. A model of machine and task heterogeneity[16] must be available to the resource allocation system, and the system must harness this awareness to overcome with the uncertainty of the HC system.

When tasks have hard deadlines, time spent executing tasks that are ultimately dropped is wasted time. This wasted time cascades down the queue of tasks, delaying the execution of other tasks, and increasing the number of missed tasks in the future—decreasing system robustness. To mitigate this, tasks with a low probability of success should not be mapped, and if they are, they should be dropped before execution[17]. If

probabilistically pruning these unlikely-to-succeed tasks yields more tasks completing on-time in oversubscribed HC systems, how do we maximize the robustness gained thereby?

To address this question, in this research, we propose a pruning mechanism [18] (as depicted in Figure 1) that is composed of two methods, namely *deferring* and *dropping*. Task deferring deals with postponing assignment of unlikely-to-succeed tasks to a next mapping event with the hope that the tasks can be mapped to a machine that provides a higher chance of success for them. Alternatively, when the system is oversubscribed, the pruning mechanism transitions to a more aggressive mode and drops the tasks that are unlikely to succeed. Before determining deferring and dropping details, we need to model the impact of task dropping on the probability of success for the tasks scheduled to execute after the dropped task. Then, we determine the appropriate probability for dropping and deferring. We propose a method to dynamically determine when the resource allocation system should transition to a more aggressive mode and engage in task dropping. We compare and analyze robustness obtained from deploying our proposed pruning mechanism against an HC system that either does not perform pruning or has a basic pruning implemented.

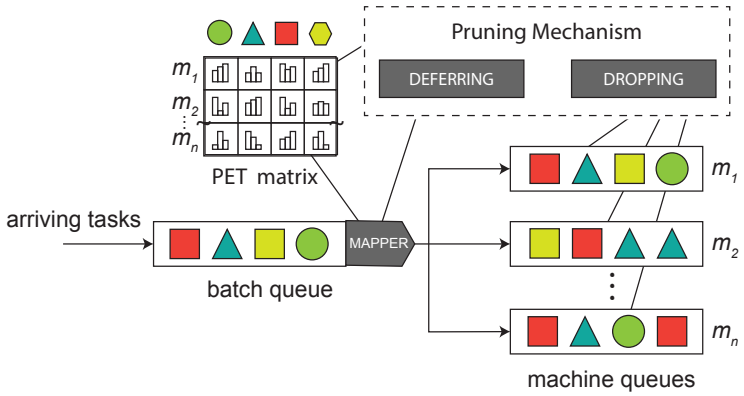


Figure 1: Pruning mechanism. Heterogeneous tasks are mapped to heterogeneous machines in batches. In each mapping, the pruner drops or defers tasks based on their probability of success.

Maximizing robustness of HC systems in terms number of tasks meeting their deadlines can potentially cause bias towards executing certain task types and affects fairness of the system. As such, we develop a mapping method to maintain fairness while maximizing robustness.

Our hypothesis is that the proposed pruning mechanism not only improves robustness of an HC system, but can impact the incurred cost and energy consumption of using resources. The former is particularly important for users who deploy heterogeneous cloud VMs [19], whereas the latter can be appealing to the administrators of High Performance Computing (HPC) systems. As such, we investigate the impact of the proposed probabilistic pruning mechanism on the incurred cost and energy consumption of using heterogeneous cloud VMs and compare it against common mapping methods. Due to generality of the pruning idea, we implement it as an independent

mechanism that can be applied to mapping heuristics of any type of (homogeneous or heterogeneous) computing system to improve its robustness.

Naively implementing the theory behind the pruning decisions imposes a significant overhead to decide the fate of a given task. As such, to make the pruning mechanism practical, we develop methods based on approximation and caching that effectively mitigate the mechanism's overhead, without major impact on the effectiveness of pruning.

In summary, the main contribution of this paper is to provide a pruning mechanism that improves robustness, cost, and energy efficiency of HC systems. More detailed contributions of this paper are as follows:

- Mathematically modeling impact of task dropping on the probability of other tasks.
- Developing a method to determine probabilistic dropping and deferring thresholds.
- Proposing a method to engage task dropping in response to oversubscription.
- Developing a pruning-aware and a fairness-aware mapping heuristic for an HC system.
- Developing a generic pruning mechanism that can be applied to existing HC systems.

Simulation results approve our hypotheses and show that the pruning mechanism can enhance robustness and the incurred cost. Importantly, the mechanism is more effective under higher oversubscription levels. This rest of this paper is organized as follows. Section 2 situates this work in relation to existing literature. Section 3 establishes the problem and describes our system model. Then, Section 5 presents theories for the probabilistic task pruning mechanisms. Section 6 summarizes the pruning mechanism and introduces two probabilistic-based mapping heuristics. Section 7 goes into detail on how to reduce the scheduling overhead when using probabilistic-based mapping heuristics and task pruning mechanisms. In Section 8, We describe baseline heuristics, along with the constraints and parameters of the experiment. Section 9 presents and analyzes the simulation results. Finally, Section 10 concludes the paper and offers direction for future works.

2. Related Works

Mapping tasks in HC systems have been shown to be an NP-complete problem [20,21]. As such, there are multiple prior efforts that achieve sub-optimal solutions. Here are some notable mentions where they are either being similar or have some influence on our work.

To model task execution times, Shestak *et al.* [13], instead of using a scalar value, lay the groundwork for the use of probability mass functions (aka PMF). The method for convolution of execution times to form completion times for a queue of tasks is established. Our work builds upon their use of PMFs and robustness measurement, while

also adding the conditions of probabilistically drop executing tasks and pending tasks. Khemka *et al.* [14] investigate resource allocation in oversubscribed heterogeneous systems. They test task utility functions based on priority, utility class, and urgency. They use a matrix with deterministic execution times, whereas we model the times probabilistically. Also, unlike our approach of probabilistically determining if a task should be dropped, their task dropping occurs only after a task's utility goes below a static threshold. In [22], Salehi *et al.* model the stochastic nature of the heterogeneous task types on heterogeneous machine types using a matrix of probability mass functions (PMFs) to improve robustness of dynamic resource allocation. A mathematical model for calculating the completion time of stochastically modeled tasks in the presence of task dropping is provided. However, Salehi *et al.* only consider dropping tasks after their deadlines have passed.

Delimitrou and Kozyrakis [23] propose Paragon which is an immediate (*i.e.,* not batch) dynamic scheduling system for heterogeneous data centers. They use singular value decomposition of historical data to classify incoming tasks based on their heterogeneity. The classifications are used in a greedy algorithm to select a list of candidate resources based on interference, and then from that, the best fit based on heterogeneity [24]. Unlike our work that considers probabilistic execution times for decision making, their mapping heuristics operates based on scalar execution times. The performance metrics are also different, as their tasks do not have deadline to consider, Paragon is only concerned about system throughput.

In [25], Li *et al.* introduce the affinity (*i.e.,* match) of heterogeneous cloud VMs to change coding of video streams. They observed that depending on their content types, video files have different performances on heterogeneous VM types. Particularly, they notice that slow-motion video contents gain from compute intensive VMs, such as GPUs, whereas fast-motion videos do not gain much from such VMs. They concluded that categorizing videos based on their content types and deploying an inconsistently heterogeneous set of cloud VMs can reduce the incurred cost of using cloud without compromising quality. In another work [2], Li *et al.* dynamically composes an inconsistently HC system to process a heterogeneous set of video streaming tasks. However, they do not consider the case of task dropping.

Malawski *et al.* [26] evaluate dynamic mapping of deadline- and cost-constrained tasks in cloud. They support dropping workflows that would result in a loss of high priority tasks completion, however, their metrics to quantify and evaluate each task's worthiness are different. Unlike our work, they focus on homogeneous cloud VMs. Tetrisched [27] is a mapping method for consistent HC systems used for YARN and MapReduce. It operates based on mixed integer linear programming and considers task execution time on different machines types. Our system uses a similar set of information to for mapping, however, it also leverages task deferring to find a better match for tasks and considers task dropping to alleviate oversubscription and improve robustness.

3. System Model

The motivation for this research comes from an HC system used for processing live video streaming services [28,29] (*e.g.,* YouTube Live and Twitch.tv [30]). In these

services, video content is initially captured in a certain format and then processed (aka transcoded) to support diverse viewers' display devices[31]. As there is no value in executing live video streaming tasks that have missed their individual deadlines, they are dropped[32] from the HC system. It has been shown that, in such a system, deploying an inconsistently HC system helps processing inconsistently heterogeneous task types (*e.g.,* tasks to change resolution and tasks to change compression standard) and ensuring an uninterrupted streaming experience[25]. Figure 1 shows an overview of the system. Tasks are queued upon arrival and are mapped to available heterogeneous machines ($m_1, m_2, ..., m_n$) in batches.

To capture the stochastic nature in execution time of each task type (*e.g.,* those arising from data-size differences in tasks), we use Probability Mass Functions (PMF). In an inconsistently HC system, the execution time PMF of different task types on different machine types are maintained in a matrix called a *Probabilistic Execution Time* (PET)[1,22]. As we consider the HC system is deployed to offer a specific service (*e.g.,* video streaming), the type of arriving task requests are limited and known. As such, the PET matrix has a limited and constant size. In practice, the PMFs of the PET matrix can be built from historic execution time information of each task type on each machine type and modeling them via a histogram in an offline manner[33]. Thus, we assume that such a PET matrix is available in our HC system.

In our system, as seen in Figure 1, heterogeneous tasks dynamically arrive into a batch queue of unmapped tasks with no prior knowledge of the timing or order. The intensity of tasks arriving to the HC system (*i.e.,* oversubscription level) also varies. To limit the compound uncertainty and maintain accuracy of mapping decisions, machines use limited-size local queues to process their assigned tasks in a first-come-first-serve (FCFS) manner (called machine queue). Such machine queues need to be large enough to not cause machine idling between mapping events. However, excessively large machine queues compounds the uncertainty in completion time that leads to imprecise mapping decisions. A mapping event occurs upon arrival of a new task or when a task gets completed. Before the mapping event, tasks that have missed their deadlines are dropped (removed) from the system. Then, the mapping event attempts to map tasks from the batch queue. This happens until either the machine queues are full, or there are no more unmapped tasks. We assume 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. It is assumed that each task is independent and executes in isolation on a machine, with no preemption and no multitasking[34,35].

To map tasks to machines, the mapper creates a temporary queue (aka, *virtual queue*) of machine-task mappings and calculates the completion time distribution of each unmapped task on heterogeneous machines, as explained in the next section.

4. Calculating Task Completion Time in the Presence of Task Dropping

Upon dropping a task in a given machine queue, the completion time PMF of those tasks behind the dropped tasks is improved. Intuitively, dropping a task, whose deadline has passed or has a low chance of success, enables the tasks behind it to begin execution sooner, thus, increasing their probability of success and subsequently, overall robustness of the HC system. Each task in queue compounds the uncertainty in the

completion time of the tasks behind it in the queue. Dropping a task excludes its PET from the convolution process, reducing the compound uncertainty as well.

The pruning mechanism we propose in this research should be able to calculate the impact of dropping a task on the probability of success (*i.e.,* success chance) of tasks behind the dropped tasks. In this section, we provide the mathematical model to calculate the completion time and probability of meeting deadline of a task located behind a dropped task.

Recall that each entry (i, j) of PET matrix is a PMF represents the *execution time* of task i's task type on a machine type j. In fact, $PET(i, j)$ is a set of impulses, denoted E_{ij}, where $e_{ij}(t)$ represents execution time probability of a single impulse at time t. Similarly, completion time PMF of task i on machine j, denoted $PCT(i, j)$, is a set of impulses, denoted C_{ij}, where $c_{ij}(t)$ is an impulse representing the probability of completing task i on machine j at time t.

Let i be a task with deadline δ_i arrives at time α and is given a start time on idle machine j. In this case, the impulses in $PET(i, j)$ are shifted by α to form its $PCT(i, j)$ [22]. Then, the success chance of task i on machine j is the probability of completing i before its deadline, denoted $p_{ij}(\delta_i)$, and is calculated based on Equation 1.

$$p_{ij}(\delta_i) = \sum_{t=\alpha}^{t \leq \delta_i} c_{ij}(t) \tag{1}$$

In case machine j is not idle (*i.e.,* it has executing or pending tasks) and task i arrives, the PCT of the last task in machine j (*i.e.,* $PCT(i-1, j)$) and $PET(i, j)$ are convolved to form $PCT(i, j)$. This new PMF accounts for execution times of all tasks ahead of task i in the machine queue j. For example, in Figure 2, an arriving task i with $\delta_i = 7$ is assigned to machine j. Then, $PET(i, j)$ is convolved with the PCT of the last task on machine queue j to form $PCT(i, j)$.

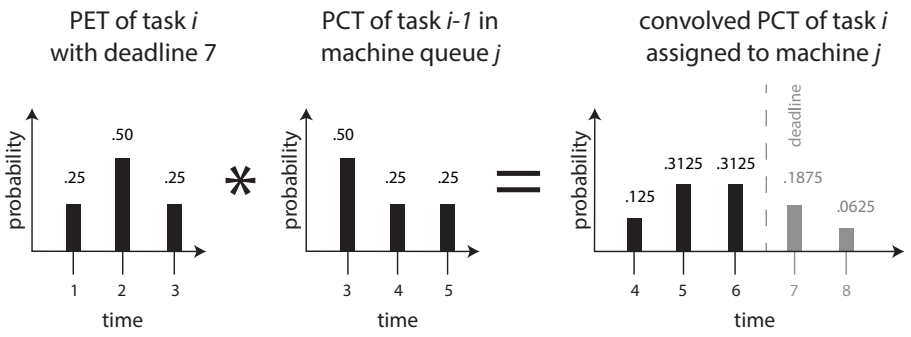


Figure 2: Probabilistic Execution Time (PET) of arriving task i is convolved with the Probabilistic Completion Time (PCT) of the last task on machine j to form $PCT(i, j)$.

The completion time impulses are generated differently based on the way task dropping is permitted in a system. Three scenarios are possible: **(A)** Task dropping is not permitted; **(B)** Only pending tasks can be dropped; and **(C)** Any task, including the executing one, can be dropped. We note that the initial idea of calculating these completion time PMFs were proposed in [22]. However, in the following, we mathematically model

and provide the closed form solution for calculating completion time PMFs. Considering the space limit, interested readers can refer to [22] for further explanations.

(A) Task dropping is not permitted, *i.e.,* when all mapped tasks must execute to completion, Equation 2 is used to calculate the impulses, denoted $c_{ij}^{NoDrop}(t)$, of C_{ij} from the convolution of $PET(i,j)$ and $PCT(i-1,j)$.

$$c_{ij}^{NoDrop}(t) = \sum_{k=1}^{k<t} [e_{ij}(k) \cdot c_{(i-1)j}^{NoDrop}(t-k)] \tag{2}$$

(B) Only pending tasks can be dropped. In this case, the impulses in $PCT(i-1,j)$ that occur after the deadline of task i are not considered in calculating $PCT(i,j)$, as that would indicate task i is dropped due to its deadline passing. Therefore, the formulation changes to reflect the impact of truncated $PCT(i-1,j)$ in the convolution process. Owing to the complexity of calculating $PCT(i,j)$, in this circumstance, we develop a helper function, denoted $f(t,k)$, as shown in Equation 3, that helps Equation 4 to discard impulses from $PCT(i-1,j) \geq \delta_i$. To calculate impulse $c_{ij}(t)$, note that if $t < \delta_i$, then $t-k < \delta_i$. In this case, Equations 4 and 3 operate the same as Equation 2. However, for cases where $t \geq \delta_i$, we use the helper Equation 3 to generate an impulse by discarding impulses of $PCT(i-1,j) \geq \delta_i$. Later, in Equation 4, we add impulses in $i-1$ that occur after δ_i to account for when task $i-1$ completes at or after δ_i.

$$f(t,k) = \begin{cases} 0, & \forall (t-k) \geq \delta_i \\ \\ e_{ij}(k) \cdot c_{(i-1)j}^{pend}(t-k), & \forall (t-k) < \delta_i \end{cases} \tag{3}$$

$$c_{ij}^{pend}(t) = \begin{cases} \sum_{k=1}^{k<t} f(t,k) + c_{(i-1)j}^{pend}(t), & \forall t \geq \delta_i \\ \\ \sum_{k=1}^{k<t} f(t,k), & \forall t < \delta_i \end{cases} \tag{4}$$

(C) All tasks (including executing one) can be dropped. In fact, in this case, the completion time impulses are obtained similar to Equation 4. However, the special case happens when $t = \delta_i$ because at this time, if task i has not completed, it is dropped. For the purposes of calculating $PCT(i,j)$ using Equation 5, $PCT(i-1,j)$ is guaranteed to be complete by its deadline. Therefore, as Equation 5 shows, all the impulses after δ_i are aggregated into the impulse at $t = \delta_i$. We should note that, the discarded impulses, *i.e.,* those of task $i-1$ that occur at or after δ_i, must be added to C_{ij}, to indicate the

probabilities that task $i-1$ completes after task i's deadline.

$$c_{ij}^{evict}(t) = \begin{cases} \sum_{k=t}^{k<\infty} c_{ij}^{pend}(k) + c_{(i-1)j}^{evict}(t), & t = \delta_i \\ c_{(i-1)j}^{evict}(t), & \forall t > \delta_i \\ \sum_{k=1}^{k<t} f(t,k), & \forall t < \delta_i \end{cases} \tag{5}$$

We note that, calculating completion time and probability of success based on the proposed theory at each mapping event poses a non-negligible overhead to the system. Therefore, in section 7, we propose methods based on approximate computing to mitigate this overhead and making pruning a practical component of a resource allocation system.

5. Maximizing Robustness via Pruning Mechanism

5.1. Overview

In the beginning of the mapping event, if the system is identified as oversubscribed, the pruning mechanism (aka pruner) examines machine queues. Beginning at the executing task (queue head), for each task in a queue, the success probability (success chance) is calculated. Tasks whose chance of success values are less than or equal to the dropping threshold are removed from the system. Then, the mapping method 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. This is in an effort to increase robustness of the system by waiting for a machine with better match to become available for processing the deferred task. To design the pruner for an HC system, three sets of questions regarding deferring and dropping operations are posed that need to be addressed.

First, a set of questions surround the probability thresholds at which tasks are dropped or deferred. How to identify these thresholds are described in Sections 5.2–5.3. A related question that arises is, should a system-level probability threshold be applied for task dropping? Or, should there be individual considerations based on the characteristics of each task? If so, what characteristics should be considered, and how should they be used in the final determination?

Second, there is the matter of when to begin task dropping, and when to cease. That is, how to dynamically determine the system is oversubscribed and transition the pruner to a more aggressive mode to drop unlikely-to-succeed tasks such that the overall system robustness is improved. The answer to this question is provided in Section 5.4.

Pruning can potentially lead to unfair scheduling across tasks types—constantly pruning compute-intensive and urgent task types in favor of other tasks to maximize the overall robustness. Hence, the *third* question is how the unfairness across task types

can be prevented? Should the system prioritize task types that have been pruned? If so, how much of a concession should be made? We address this question in Section 6.

5.2. Determining Task Dropping Probability

5.2.1. Dynamic Per-Task Dropping Probability Threshold

At its simplest, the task dropper can apply uniform dropping threshold for all tasks in a machine queue. However, a deeper analysis tells us that not all tasks have the same effects on the probability of on-time completion for the tasks behind them in queue. This difference can be taken into account to make the best decision about which tasks should stay and which are dropped.

In addition to determining task's chance of success, other features of completion time PMF can be valuable in making decisions about probabilistic task dropping. We identify two task-level characteristics that further influence the chance of success of tasks located behind a given task i: (A) the position of task i in machine queue, and (B) the shape (*i.e.,* skewness) of completion time PMF of task i.

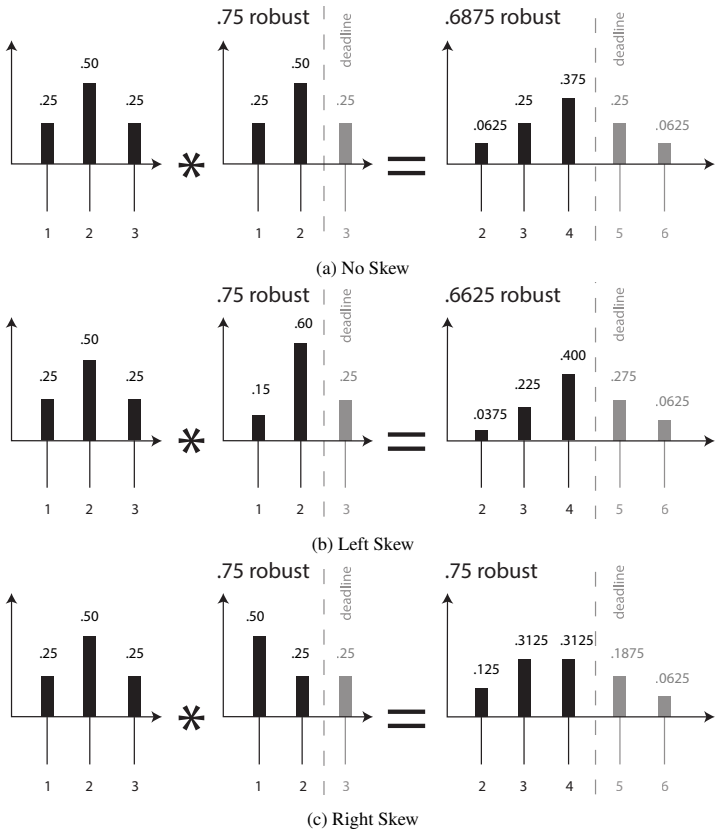
In fact, the closer a task is to execution (*i.e.,* to the head of machine queue), the more tasks are affected by its completion time. For instance, with a machine queue size of six, an executing task affects the completion time of five tasks queued behind it, where the execution time of a task at the end of the queue affects no tasks. Therefore, the system should apply a higher dropping threshold for tasks close to queue head.

Skewness is defined as the measure of asymmetry in a probability distribution and is calculated based on Equation 6, as explained in [36]. In this equation, N is the sample size of a given PMF, Y_i is an observation, $\bar{Y}$ is the mean of observations, and σ is the standard deviation of the observations. A negative skewness value means the tail is on the left side of a distribution whereas a positive value means that the tale is on the right side. Generally, $|S| \geq 1$ is considered highly skewed, thus, we define s as bounded skewness and we have $-1 \leq s \leq 1$.

$$S = \frac{\sqrt{N(N-1)}}{N-2} \times \frac{\sum_{i=1}^{n} (Y_i - \bar{Y})^3 / N}{\sigma^3} \tag{6}$$

A negatively skewed PMF has the majority of probability occurring on the right side of PMF. Alternatively, because the bulk of a probability is biased to the left side of a PMF, a positive skew implies that a task is completed sooner than later. The middle PMFs in Figure 3 each represents a completion time with a success chance of 0.75, however, they show different types of skewness. Using this information, we can see that two tasks with the same success chance can have different impacts on the success chance of tasks behind them in queue. Tasks that are more likely to complete sooner (*i.e.,* positive skewness) propagate that positive impact to tasks behind them in queue. The opposite is true for negatively skewed tasks. Reasonably, we can favor tasks with positive skewness in task dropping. Figure 3 shows the effects of different types of skews on the completion times of tasks behind them in queue. Subfigure 3b shows the negative effects of negative skew whereas Subfigure 3c shows the positive effect of positive skew on the success chance of the next task in the queue.

Using the skewness and queue position, the system can adjust a base dropping threshold dynamically, for each task in a machine queue. The adjusted dropping thresh-



(a) No Skew

(b) Left Skew

(c) Right Skew

Figure 3: Demonstration of effect of task i's skewness on completion time PMF of task $i+1$ (right-most PMFs) with a deadline 5 ($\delta_{i+1} = 5$). The left-most PMFs show execution time PMF of task $i+1$ and the middle ones show completion time PMF of task i ($\delta_i = 3$).

old for a given task i, denoted ϕ_i, is calculated based on Equation 7. To favor tasks with positively skewed completion time PMF, we negate the skewness (s_i). To account for position of task i in machine queue, denoted κ_i, we divide the negated skewness by the position. Addition of 1 is just to avoid division by zero and ρ is a parameter to scale the adjusted dropping threshold. Ideally, this will allow more tasks to complete before their deadline, leading to a higher robustness in an HC system.

$$\phi_i = \frac{-s_i \cdot \rho}{\kappa_i + 1} \tag{7}$$

This dynamic adjustment of the probability is done only in the dropping stage of the pruner. When it comes to deferring tasks, the task position 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.

5.3. Determining Task Deferring Probability

We discussed that any task that has a chance of success lower than its specified dropping threshold has too low of a success chance to warrant risking of allocating it to a resource. The optimal deferring value, however, is applied to unmapped tasks in the batch queue and should vary based on the workload characteristics. In fact, deferring threshold acts as a throttle that controls the flow of incoming tasks to the HC system. In one hand, a too-high deferring threshold ensures that available machine queue slots are reserved only for tasks with a high chance of success, but can lead to resource under-utilization by leaving the computing resource idle. On the other hand, a too-low deferring threshold allows tasks, potentially with low chance of success, to fill the machine queue slots—preventing the mapping of high-chance incoming tasks. To avoid such scenarios, an appropriate deferring threshold should be dynamically determined based on the characteristics of the workload in the system. In the rest of this section, we describe our approach to dynamically adjust the deferring threshold based on the workload characteristics.

Selective factor (denoted Δ) is defined as the ratio of the number of tasks in batch queue (waiting to be mapped) to the number of empty slots in machine queues. A high selective factor indicates that there are many unmapped tasks, but not enough machine queue slots to accommodate them. In this scenario, task mapping should be more selective and task dropping should be more aggressive to free up machine queue slots for a better-suited tasks that are waiting to be mapped.

Let υ be the deferring threshold in an HC system with b unmapped tasks in its batch queue. A *competent task* is a task whose maximum success chance across all machines is higher than the υ. That is, competent tasks are those that are not deferred because they have decent chance of success. Accordingly, *task competency level* (denoted Γ) in a batch queue is defined as the ratio of the number of competent tasks to the total number of unmapped tasks and is calculated based on Equation 8. High task competency level (close to 1) implies a high percentage of tasks in the batch queue are qualified for mapping, but are not mapped due to inadequate slots in the machine queue. In this case, the deferring threshold should be increased, so that only the highly competent tasks are considered for mapping. Conversely, a low task competency level

can be an indication that the task deferring threshold is set too high, *i.e.,* the system is too selective, such that the majority of the tasks cannot pass the deferring threshold.

$$\Gamma = \frac{1}{b}\sum_{i=1}^{b}\begin{cases}0, & max(p_{ij}(\delta_i)) < \upsilon | j \in \{0..m\}\\ 1, & max(p_{ij}(\delta_i)) \geq \upsilon | j \in \{0..m\}\end{cases} \tag{8}$$

5.3.1. Instantaneous Robustness

Instantaneous robustness at a given time is defined as the average of chance of success for all tasks exist in the system. Let m be the number of machine queues, q the number of queue slots in each machine queue, and $p_{ij}(\delta_i)$ is the chance of meeting the deadline of task i in the machine queue j. Then, instantaneous robustness is calculated based on Equation 9. Our hypothesis is that maintaining a high instantaneous robustness leads to a high level of overall system robustness. As such, instantaneous robustness can act as a performance indicator for task deferral and mapping heuristics. The system should aim to maintain the high instantaneous robustness level and avoid task mappings that reduce the instantaneous robustness.

$$\psi = \frac{1}{m \cdot q}\sum_{j=1}^{m}\sum_{i=1}^{q} p_{ij}(\delta_i) \tag{9}$$

5.3.2. Deferring Probability Threshold

When the system is not heavily oversubscribed and there are more empty slots in the machine queue than tasks in the batch queue (*i.e.,* $\Delta < 1$), the new deferring threshold (υ_n) can be reduced from its current value (υ_c) to allow more task mappings. Alternatively, when there are more tasks to map than the number of available slots (*i.e.,* $\Delta > 1$), we act based on the competency level (Γ). If no task is passing its deferring threshold (*i.e.,* $\Gamma = 0$), it means that the deferring threshold is high and has to be reduced. Otherwise, in the case of oversubscription, we set the deferring threshold to a value near the instantaneous robustness value. The vale of θ is a constant to adjust the deferring probability threshold. Equation 10 formally expresses the way deferring probability threshold is dynamically calculated.

$$\upsilon_n = \begin{cases}\upsilon_c - \theta, & \Delta < 1\\ \psi - \theta, & \Delta \geq 1, \Gamma \neq 0\\ \upsilon_c - \theta, & \Delta \geq 1, \Gamma = 0\end{cases} \tag{10}$$

5.4. Aggressive Pruning by Dynamically Engaging Task Dropping

To maximize robustness of the system, the aggression of the pruning mechanism has to be dynamically adjusted in reaction to the level of oversubscription in the HC system. The pruning mechanism considers the number of tasks missed that their deadlines since the past mapping event as an indicator of the oversubscription level in the system. We use the identified oversubscription level as a *toggle* that transitions the

pruner to task dropping mode. However, in this case, the pruner can potentially toggle to dropping mode as a result of an acute spike in task-arrival, and not a sustained oversubscription state.

To judge the oversubscription state in the system, the pruner operates based on a moving weighted average number of tasks that missed their deadlines during the past mapping events. Let d_τ the oversubscription level of the HC system at mapping event τ; and μ_τ 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. The oversubscription level is the calculated based on Equation 11. In the experiment section, we analyze the impact of *lambda* and determine an appropriate value for it.

$$d_\tau = \mu_\tau \cdot \lambda + d_{\tau-1} \cdot (1-\lambda) \tag{11}$$

Another potential concern is minor fluctuations about the toggle switching the dropping off and then back on. We employ a Schmitt Trigger[37] to prevent minor fluctuations around dropping toggle. We set separate on and off values for the dropping toggle. Based on our initial experiments, we determined the Schmitt Trigger to have 20% separation between the on and off values. For instance, if oversubscription level two or higher signals starting dropping, oversubscription value 1.6 or lower signals stopping it.

6. Pruning Mechanism as Module of a Resource Allocation System

6.1. Overview

In this section, with the goal of maximizing the system robustness, theories presented in Sections 4 and 5 are leveraged to design a pruning mechanism as a module of resource allocation system that can work with any mapping heuristic. Then, two probabilistic mapping heuristics, called Pruning Aware Mapper (PAM) and Fair Pruning Aware Mapper (PAMF) are proposed to work along with the pruning mechanism.

6.2. Task Pruning Mechanism

The overall architecture of the pruning mechanism is shown in Figure 4. The *Accounting* module receives meta data (*e.g.,* tasks' deadline, PET, and PCT) from the resource allocation system. The meta-data are available for other components to utilize. The *Toggle* module (in either the default or Schmitt trigger configuration) uses the collected information to measure the oversubscription level of the HC system. It then decides whether or not it is beneficial to engage the "task dropping".

With the goal of maximizing the robustness, the *Pruner* module enacts the the dropping and deferring sub-modules to prune tasks whose chance of success is lower than the thresholds specified in the *Pruning Configuration*. Moreover, dropping and deferring thresholds are dynamically adjusted during each mapping event to maximize the system robustness. To this end, the *Dropping Threshold Estimator* modifies the dropping threshold based on each task's skewness and position in the machine queue. Furthermore, The *Deferring Threshold Estimator* module adjusts the deferring threshold based on the oversubscription level.

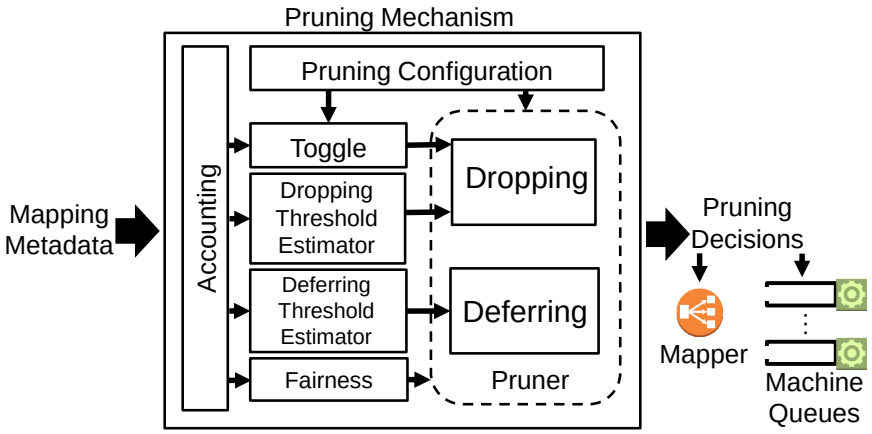


Figure 4: Components of pruning mechanism. Inputs are mapping metadata and outputs are pruning decisions to apply on mapper (task deferring) or machine queues (task dropping). The pruning system is packaged as module in resource allocation systems to function in conjunction with mapping heuristic.

The *Fairness* module is employed to avoid unfair pruning mechanism. This module detects the suffering task types (*i.e.,* those that are consistently dropped) and adjusts the pruner to prevent task types being unfairly pruned. The Fairness module is elaborated in Section 6.3.2. Output of the pruning mechanism is its decision that can be either task dropping (applied on machine queues) or task pruning (applied on unmapped queue).

As mentioned earlier, the pruning mechanism is pluggable, that is, it can be added to any mapping heuristic. In the next part, we explain plugging the pruning mechanism to two new probabilistic-base mapping heuristics, namely PAM and PAMF.

6.3. Mapping Heuristics with Pruning Mechanism

In this part, two mapping heuristics that work in conjunction with the pruning mechanism are developed. First, a heuristic called Pruning Aware Mapper (PAM) leverages the chance of success to probabilistically maximize the robustness of the system. In this scope, the robustness of the system is defined as the number of tasks completed on time during the study time. However, only considering the chance of success to maximize the robustness can result in unfair task type completion. As a result, a second mapping heuristics is proposed to achieve maximum robustness balanced with fairness across task types. The heuristics occur in two phases. In the first phase, for each task, a machine that has the best affinity is determined and a task-machine pair constructed. PAM considers task-machine affinity implicitly via taking the chance of success of a task into consideration. Then, the best task-machine pair is selected for mapping and that task is assigned to its paired machine. Note that the pruning mechanism can be plugged in mapping heuristics to maximize the system robustness. The pruning occurs in two steps. First, prior to any mapping decision, the pruner performs task-dropping on machine queues. Next, tasks in the batch queue with chance

of success lower than the deferring threshold are deferred, leaving the deferred tasks to remain in the batch queue.

6.3.1. Pruning Aware Mapper (PAM)

In PAM, maximizing the system robustness happens by maximizing each task's chance of success. To this end, the PET matrix is used to determine the chance of success for each task.

Based on our prior observations, the machine offering highest chance of success is selected for constructing the task-machine pair during the first phase of the PAM heuristic. Then, in the second phase, the pair with lowest completion time is selected for mapping. In this way, the system prefers to map tasks having both high chance of success and short execution time.

6.3.2. Fair Pruning Aware Mapper (PAMF)

Probabilistic task pruning potentially favors task types with shorter execution times, resulting in unfairness. This is because shorter tasks usually have a higher probability of completion within their deadlines. PAMF is designed to mitigate such unfair task pruning. In this mapping heuristic, thresholds (dropping and deferring) are adjusted for task types unfairly treated.

We define a *sufferage value* at mapping event e for each task type f, denoted ε_{ef}, that determines how much to decrease (*i.e.,* relax) the base pruning threshold. Note that we define 0 as no sufferage. We define *fairness factor* (denoted ϑ) as a constant value across all task types in a given HC system by which we change sufferage value of task types. This fairness factor denotes how quickly any task's sufferage value changes in response to missing a deadline. A high factor results in large relaxation of probabilistic requirements. Updating the sufferage value occurs upon completion of a task in the system.

A successful completion of a task of type f in mapping event e results in lowering the sufferage value of task type f by the fairness factor, *i.e.,* $\varepsilon_{ef} = \varepsilon_{(e-1)f} - \vartheta$, whereas for an unsuccessful task we add the fairness factor, *i.e.,* $\varepsilon_{ef} = \varepsilon_{(e-1)f} + \vartheta$. Note that we limit sufferage values (ε_{ef}) to be between 0 to 100%. The mapping heuristic determines the fair pruning threshold for a given task type f at a mapping event e by subtracting the sufferage value from the base pruning threshold.

This updated pruning threshold enables PAMF creates a more fair distribution of completed tasks by protecting tasks of unfairly-treated types from pruning. Once we update pruning thresholds for suffered task types, the rest of PAMF functions as PAM.

7. Practicality of the Pruning Mechanism

One concern when considering the deployment of probabilistic approaches in the mapping of tasks is the extra computational overhead. Repeated convolutions put strains on the machine that handle task mapping, especially when tasks are small and come in large numbers. To ensure a probabilistic task pruning mechanism and PAM are real-world practical, this section describes some techniques that can be used to mitigate the scheduling overhead.

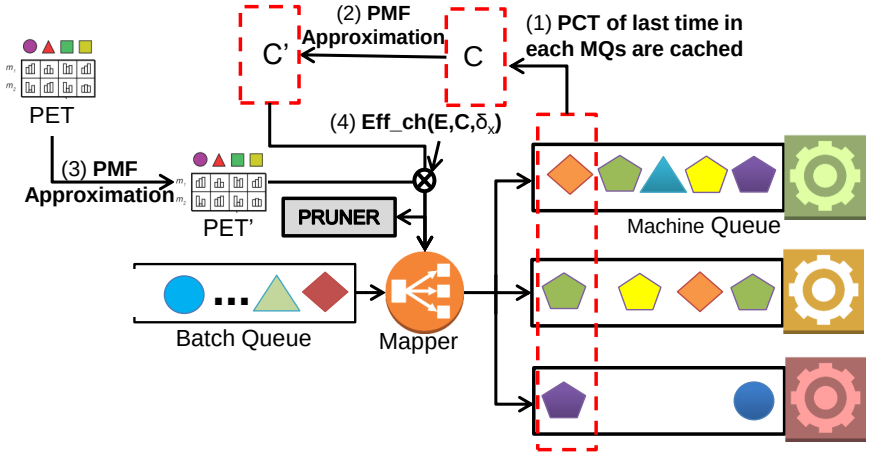


Figure 5: Overview of optimization strategies, (1) PCT of last task in the machine queue is predetermined before mapping event, (2) and (3) perform PMF compaction (approximation) of last task in machine queue's PCT and arrival task's PET respectively, (4) Chance of success can be calculated by an algorithm with memoization without a complete convolution.

7.1. Macro-level Memoization to Reduce Redundant Calculations

During the first and second phase of PAM (and most probabilistic mapping heuristics), each and every unmapped task's execution time PMF (PET) are convolved with the PCT of the last tasks of each machine queue. Finding PCT is a chain convolution process that starts from the head of the machine queue. Supposing N tasks in each of the M machine queues, to find the PCT of B unmapped tasks, there can be up to $B \cdot N \cdot M$ convolutions. Therefore it is recommended to cache the PCT of the last task in each machine queue before the mapping event to remove the repetitive convolutions on the machine queue (this is shown as step (1) in Figure 5). This reduces the number of convolutions to $B \cdot M + N \cdot M$ where $N \cdot M$ part is cached at the beginning of the mapping event. Note that this caching is only valid for a single mapping event. Once the current time passes, this cache is no longer valid.

Once the PCT of the last task in the machine queue is calculated, it is also possible to perform PMF approximation on the PCT. Which will be explained in the next part.

7.2. Approximation to Reduce Convolutions Overhead

It is well known that the convolution process can impose a significant computation due to the sheer number of impulses that form the PCT after a chain of compound convolutions. Therefore, to alleviate scheduling overhead, some dynamic programming and approximating techniques are utilized to reduce the time spent in PMF convolution process. In this part, we first introduce a procedure to reduce the number of impulses. We then propose a procedure to replace the last step of the convolution process in a probabilistic mapping heuristic: convolving the PCT of the last task in machine queue and the PET of each unmapped task. Due to the size of PCTs convolved from the machine queue, a large number of unmapped tasks can impose a significant computational overhead, and warrants a customized algorithm.

7.2.1. PMF Approximation

Convolution process time relates directly to the number of impulses in the PMF. In the case that the PMF is too finely detailed, convolution can be a burden with the calculation of many small impulses. Due to high uncertainty in a heterogeneous computing system, the extra resolution may not yield significantly better decision making. We can therefore, in some calculations, use an approximate PMF which has lower number of impulses than a detailed original PMF. The approximate PMF can be created by combining multiple impulses in a specific range together as shown in Figure 6. In the approximation process, in the case that we know the range of minimum and maximum time impulses to keep in the distribution, the distribution can also be cropped to the specified range. An example of the case where the relevant range is known is when the impulses of last task in the machine queue's PCTs (step (2) in Figure 5) are being approximated. The maximum time can be set as the longest deadline of the entire unmapped task without effecting the chance of success measurement in the task mapping process.

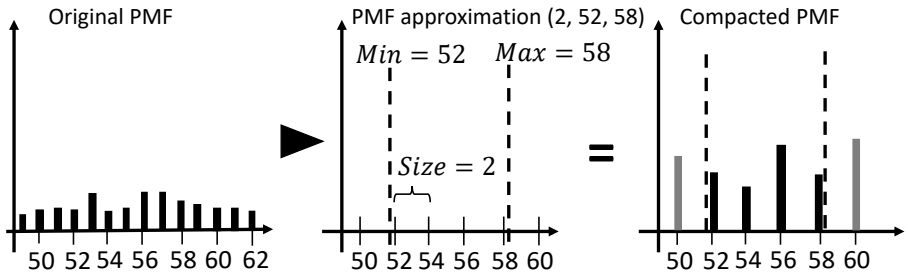


Figure 6: An example of impulse approximation process with bucket size of two and minimum and maximum range set to 52, and 58. Impulses are grouped and combined in 2 time unit interval in the specific range. All impulses that are more than specified max or less than specified min are combined together.

7.2.2. Micro-level Memoization to Reduce Convolution Overhead

Probabilistic mapping heuristics require calculations of unmapped tasks' chances of success to make mapping decisions. To find these probabilities in a straightforward way, we first compute each unmapped task's PCT (completion time PMF) by convolving the unmapped task's PET against the PCT of the last task of each machine queue (which can be memorized and approximated, as mentioned earlier). Then we measure the resulting PCT against the task's deadline to calculate the chance of success. The PCT of each unmapped task is only calculated to find the potential mapping probability and is not reused. Therefore, to speed up the process, instead of calculating an unmapped task's PCT before measuring its chance of success, we propose a process that directly calculates the success chance from two distributions (PET of the unmapped task and PCT of the last task in a machine queue) without creating a full PCT of the unmapped task first.

Assuming that both distributions' impulses can be iterate through in a sorted order from the earliest time to the latest time (*i.e.,* impulses are sorted by their time).

The procedure virtually performs a partial convolution on only pairs of impulses that together represent the resulting time less than the specified deadline. And since the impulses are sorted in order, some partial results between the iterations ($memo_c$ in the algorithm) can be memorized.

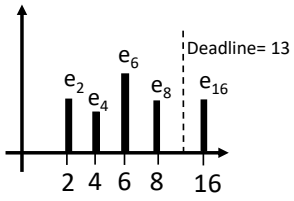
Algorithm 1: Efficient Calculation of Probability of Success for Task x on Machine j.

Input: $E \leftarrow$ PET of an unmappped task x
Input: $C \leftarrow$ PCT of the last task in the machine queue j
Input: $\delta_x \leftarrow$ Deadline of task x
Result: Chance of convolved distribution finish before deadline

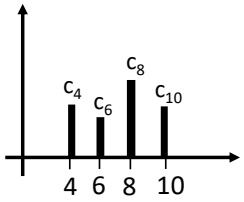
```
1  e ← First impulse of E;
2  c ← First impulse of C;
3  k ← e;
4  p_xj ← 0 ;
5  memo_c ← 0 ;
6  while time(k+1) < δ_x do
7      k ← next impulse after k ;
8  while k ≠ e do
9      while time(c)+time(k) < δ_x do
10         memo_c ← memo_c + chance(c) ;
11         c ← next impulse after c;
12     p_xj ← p_xj + memo_c · chance(k);
13     k ← previous impulse before k;
14 Return p_xj;
```

Algorithm 1 takes input as the distribution E (PET of an unmapped task x), distribution C (cached PCT of the last task in a machine queue j), and the deadline δ_x for measuring the chance of success. $chance(c)$ signifies the probability associated with the impulse c in the distribution C, and $time(c)$ signifies the time value associated with the impulse c. Line 6-7 finds the last impulse of Distribution E that is less than δ_x. Line 8 and 13 iterate back from the impulse found from line 7 back to the first impulse. Line 9 to 11 combine all chance from C's impulses that when pairing them with a specific impulse of distribution E from line 8 still provides a combined time of less than δ_x. Finally, Line 12 sum the multiplication of impulse k from distribution E (line 8) and the combined chance from lines 9-11. Note that the $memo_c$ and impulse c from line 10 and 11 are not reset on any iteration. The value always carry over from one iteration to the next. The final result is the chance of meeting the deadline δ_x as if distribution E and distribution C are convolved together.

A simplified example of the Algorithm 1 is provided in Figure 7. In this example, task x's deadline is at the time 13, the procedure runs through distribution E and C in 4 iterations where it considers one of E's impulse per iteration. Note that the impulse that has the time 16 is ignored as it is greater than the deadline. During each iteration, it considers C's impulses that can combine with the targeted e impulse and still meeting the deadline. Some partial results are carried over from prior iterations. The right most column is the p_{xj} value after each iteration. And the bottom right most cell contains



E = PET of an unmapped task



C = Cached PCT of last task of a machine queue

Iteration	Time impulse from E	Time impulses from B that meet the criteria	Chance of impulse from E	Chances of impulses from C that meet the criteria	Sum of the probability to meeting deadline
1	Impulse 8	Impulse 4	e_8	c_4	$=e_8.(c_4)$
2	Impulse 6	Impulse *(4)* , 6	e_6	*(c_4)* + c_6	$=e_8.(c_4) + e_6.(c_4 + c_6)$
3	Impulse 4	Impulse *(4 , 6)* , 8	e_4	*$(c_4 + c_6)$* + c_8	$=e_8.(c_4) + e_6.(c_4 + c_6) + e_4.(c_4 + c_6 + c_8)$
4	Impulse 2	Impulse *(4, 6 , 8)* , 10	e_2	*$(c_4+c_6+c_8)$* + c_{10}	$=e_8.(c_4) + e_6.(c_4 + c_6) + e_4.(c_4 + c_6 + c_8) + e_2.(c_4 + c_6 + c_8 + c_{10})$

Figure 7: A simplified example of Procedure 1. E is the PET of an unmapped task, C is PCT of the last task in a machine queue. The table goes through each iteration from start to finish. Notions in *Italic* are carried from their prior iteration. The carry over notion on the two right most column are stored as a scalar value denoted $memo_c$ and p_{xj} in the algorithm, respectively.

all the values that constitute the chance of meeting the deadline as if distribution E and distribution C are convolved together.

Supposing distribution E has p impulses and distribution C has r impulses, the straightforward convolution requires at least $p \cdot r$ multiplications. Measuring the chance of success also requires another run through combined impulses. Algorithm 1 is a combination of the two process together. And it loops through distribution E at most twice and distribution C at most once (rather than p times). The multiplication happens at most p time. Hence we reduce the time complexity from $p \cdot r$ to $2 \cdot p + r$, significantly speeding up the measurement of the chance of success in probabilistic-based mapping heuristics.

8. Experimental Setup

8.1. Overview

To conduct a comprehensive performance evaluation, we simulate a computing system with eight inconsistently heterogeneous machines (*i.e.*, $M = 8$). To generate the probabilistic execution time PMFs (PET), the mean execution time results from twelve

SPECint benchmarks on a set of eight bare-metal machines[1] were determined. These mean execution times for each benchmark on each system formed the mean values for our 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). 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 matrix which remains constant across all of our experiments. We note that other statistical methods can be explored to learn and tweak PMF distributions in an online manner.

8.2. Generating Workload

Our simulation is of a finite span of time units, starting and ending in a state where the system is idle. As the system comes online, 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 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 other workload investigations[15,14], 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.

Recall that we consider each task to have an individual hard deadline and it has to be dropped once the deadline is missed. For a given task i, the deadline is calculated as $\delta_i = arr_i + avg_i + (\beta \cdot avg_{all})$, where arr_i is the arrival time, avg_i is the mean execution time for that task type (range from 50 to 200 ms) , β is a slack coefficient, and avg_{all} is the mean of all task type's execution. This slack allows for the tasks to have a chance of completion in an oversubscribed system.

8.3. Baseline Mapping Heuristics

8.3.1. MinCompletion-MinCompletion (MM)

This heuristic has been extensively used in the literature[38,39,40,22]. In the first phase of the 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 free slot, the provisional mapping pairs

[1]The 8 machines are: Dell Precision 380 3 GHz Pentium Extreme, Apple iMac 2 GHz Intel Core Duo, Apple XServe 2 GHz Intel Core Duo, IBM System X 3455 AMD Opteron 2347, Shuttle SN25P AMD Athlon 64 FX-60, IBM System P 570 4.7 GHz, SunFire 3800, and IBM BladeCenter HS21XM.

are examined to find the machine-task pair with the minimum completion time, and the assignment is made to the machine queues. The process repeats itself until all machine queues are full, or until the batch queue is exhausted.

8.3.2. MinCompletion-Soonest Deadline (MSD)

Phase one is as in MM. Phase two selects the tasks for each machine with the soonest deadline. In the event of a tie, the task with the minimum expected completion time is selected. As with MM, after each machine with an available 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 unmapped task queue is empty.

8.3.3. MinCompletion-MaxUrgency (MMU)

Urgency of task i on machine j is defined as $U_{ij} = 1/(\delta_i - \mathbb{E}[C_{ij}])$, where δ_i is the deadline of task i, $\mathbb{E}[C_{ij}]$ is the expected completion time of task i on machine j.

Phase one of MMU is the same as MM. 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 batch queue is empty, or until the virtual machine queues are full.

8.3.4. Max Ontime Completions (MOC)

The MOC heuristic was developed in[22]. It uses the PET matrix to calculate robustness of task-machine mappings. The first mapping phase finds, for each task, the machine offering the highest robustness value. The culling phase clears the virtual queue of any tasks that fail to meet a pre-defined (30%) robustness threshold. 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 batch queue are mapped or dropped, or until the virtual machine queues are full.

9. Performance Evaluation

9.1. Overview

A series of simulations were run using the Louisiana Optical Network Infrastructure (LONI) Queen Bee 2 HPC system[8]. For each set of tests, for each examined parameter, 30 workload trials were 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 reported. The arrival rates are listed in terms of number of tasks per time unit.

Each experiment is a set of 30 workload trials, consisting of 1200 tasks per trial. Each of the experiments investigates high levels of oversubscription where few tasks complete successfully using baseline heuristics. Due to frequent task mapping events, each machine in the HC system has a machine-queue size of three, counting the executing task and the dropping toggle as one task. We also evaluated the system with the size of machine-queue equals to six, and the results were consistent with the presented ones. For each of the experiments, unless otherwise noted, the performance metric

(and the vertical axis) is the percentage of tasks completed before their deadline (*i.e.,* overall robustness).

9.2. Dynamic Engagement of Probabilistic Task Dropping

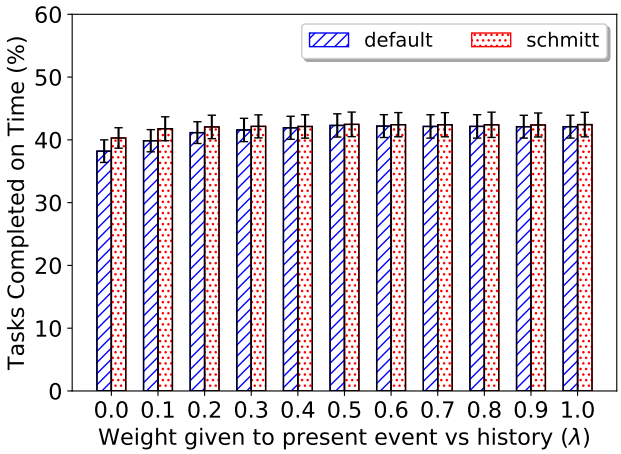


Figure 8: Impact of historical oversubscription observations and Schmitt Trigger on determining oversubscription level of HC system. Horizental axis represents the value of λ coefficient in Equation 11.

In this experiment, our aim is to appropriately measure the oversubscription level (see Equation 11) by determining the weight that should be assigned to the number of deadlines missed in the recent mapping event versus the previous values of the oversubscription level. We also evaluate the impact of using Schmitt Trigger as opposed to using a single threshold for dynamic engagement of task dropping. This experiment was conducted under 25k tasks arriving to the system.

Figure 8 shows that by assigning a higher weight to the number of dropped tasks in the most recent mapping event, the overall robustness of the system is increased from 39.9% to 42.5%. This is due in part to the steady nature of task-arrival in our workload trials with only few sudden spikes. While the maximum robustness is reached with $\lambda = 0.9$ with Schmitt Trigger, enabling Schmitt Trigger alone make the bigger difference than setting the λ to an optimal value. The system robustness of $\lambda = 1$ is close enough to the result with $\lambda = 0.9$, while ignoring the history tracking altogether which can incur less scheduling overhead.

We can conclude that under high oversubscription levels, the best results come from taking immediate action when tasks miss their deadlines, and then a steady application of probabilistic task dropping until the situation is decidedly controlled (*i.e.,* reaching the lower bound of Schmitt Trigger).

9.3. Evaluating the Mutual Impacts of Deferring and Dropping Thresholds

The goal of this experiment is two-fold: First, it identifies the impact of choosing a proper initial dropping threshold; Second, it evaluates the impact of deferring threshold on effectiveness of the dropping. For that purpose, we disable dynamic deferring

threshold and set it statically. Note that, if the workload characteristics is known, it can be helpful to set deferring threshold statically to reduce the pruning overhead.

A static deferring threshold has to be designated greater than the dropping threshold. Otherwise, a task can be dropped immediately, once it is mapped. Accordingly, to conduct this evaluation, we add a gap value to the initial dropping threshold (*e.g.,* a dropping threshold of 50% would require at least 55% robustness to map a task to a machine). Three dropping thresholds (25%, 50%, and 75%) are examined and the 5% gap is increased until the deferring threshold reaches 90%. The results, shown in Figure 9, are generated from a workload with 25k tasks.

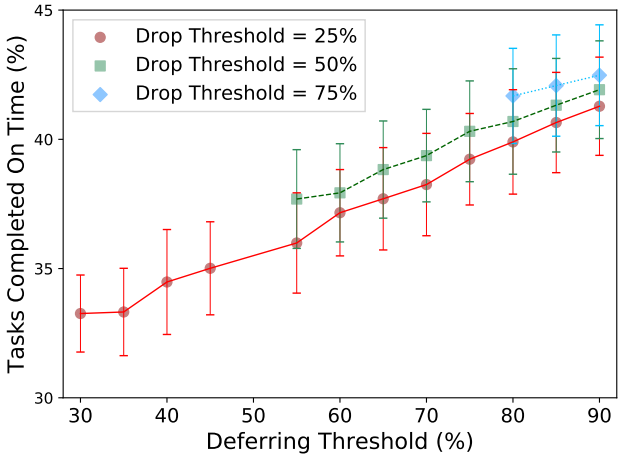
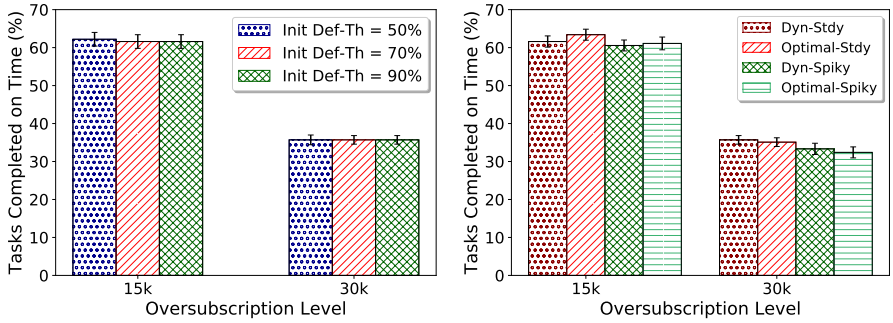


Figure 9: Impact of deferring and dropping thresholds on the system robustness. Dropping threshold is denoted by line type and color.

Figure 9 validates the experiment assumption, by showing that using a higher deferring threshold leads to higher system robustness. In addition, we observe that if the deferring threshold is chosen high enough, deferring operation prevails dropping and diminishes its influence on the system robustness. Specifically, if we choose deferring threshold at 90%, we obtain a similar system robustness, regardless of the intitial dropping threshold value. It is noteworthy that a higher dropping threshold influences the incurred cost of using an HC system, because they prevent wasting time processing unlikely-to-succeed tasks that have been mapped to the system. Based on the experiment, in the rest of evaluations, initial dropping threshold 50% is used.

9.4. Evaluating the Impact of Deferring on Various Types of Workloads

In this experiment, our goal is to evaluate effectiveness of the dynamic deferring threshold adjustment in various scenarios. First, we examine if the initial value of dynamic deferring threshold matters for the ultimate system robustness. For that purpose, we vary the initial deferring probability threshold and study the system robustness using PAM heuristic. Specifically, we examined initial deferring thresholds (shown as *Init Def-th*) to 50%, 70%, and 90%. Results of the experiment in Figure 10a shows



(a) Impact of initial deferring thresholds

(b) Impact of pruning on steady and spiky workloads

Figure 10: Evaluating the impact of dynamic deferring probability on the system robustness on the system with oversubscription level of 15k and 30k

. (a) The effect of choosing different initial deferring thresholds on the system robustness. (b) Comparing dynamic deferring probability threshold (denoted with *Dyn-* prefix) against the best statically-determined deferring threshold (denoted with *Optimal-* prefix) for both steady (stdy) and spiky workloads.

that, as the system adjusts the deferring probability threshold dynamically, the initial deferring threshold does not make a difference to the final system robustness values and they are nearly identical, regardless of the initial deferring threshold.

Second, we compare the performance of PAM when it is geared to a pruning mechanism that uses dynamic deferring threshold against when the pruning mechanism is set to the best experimentally-found deferring threshold value. Note that, in the latter case, the deferring threshold is static and does not change throughout the experiment. Also, note that the deferring threshold is the best for the examined workload and the best value might be different for other workloads. To assure the applicability of the analysis to any workload, we study two types of arriving workloads: (A) Steady task arrival rate (shown as *stdy* in the experiment); and (B) Varying arrival rate (shown as *spiky* in the experiment). The varying arrival rate workload has the same number of total tasks arriving to the system as the steady one, but with burst task arrival periods. That is, within each time interval, the task arrival rate switches between on-peak (*i.e.,* high arrival rate) and off-peak (*i.e.,* low arrival rate) periods. In summary, by combining static and dynamic deferring threshold and steady or spiky workloads, we evaluate four cases, shown as *dyn-stdy, best-stdy, dyn-spiky,* and *best-spiky*, in Figure 10b.

Figure 10b expresses that, in both steady and spiky workloads, the dynamic threshold provides almost the same robustness as to the best-known static deferring threshold. In addition, comparison between steady and spiky workload reveals that the pruning mechanism does not suffer significantly from the uncertainties in task arrival rate. That is, the system shows to be robust against the uncertainties in task arrival rate.

9.5. Evaluating the Impact of Fairness Factor

Our aim is to study if PAMF heuristic (see Section 5) alleviates unfairness. We test the system using a fairness factor ranging from 0% (*i.e.,* no fairness adjustment) to

25%. Recall that this fairness factor is the amount by which we modify the sufferage value for each task type. The sufferage value for a given task type at a given mapping event is subtracted from the required threshold, in an effort to promote fairness in completions amongst task types. For each fairness factor, we report: (A) The variance in percentage of each task type completing on time. The objective is to minimize the variance among these. (B) The overall robustness of the system, to understand the robustness we have to compromise to attain fairness. Robustness value is noted above each bar in Figure 11. We tested oversubscription level of 25k and 30k tasks.

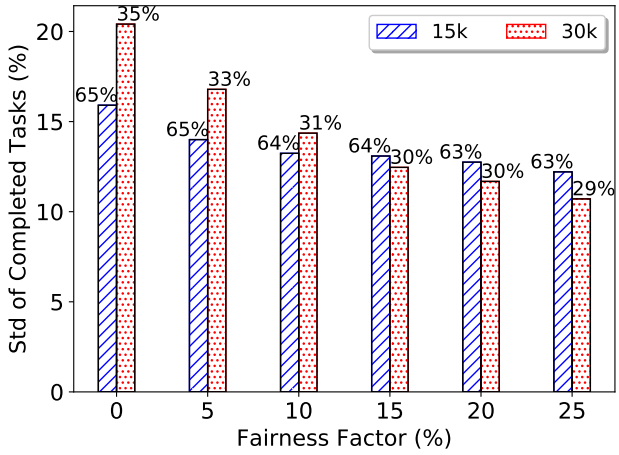


Figure 11: Evaluating fairness and robustness on the system with 15k and 30k oversubscription level . Horizontal axis shows fairness factor modifier to the sufferage value. Vertical axis is the standard deviation of completed task types. Values above bars show robustness.

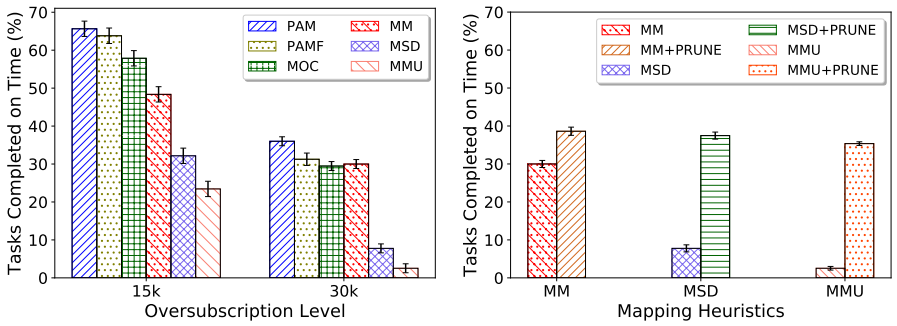
Figure 11 shows that significant improvement in fairness can be attained at the cost of compromising robustness. In the case of 15k oversubscription level, we observe that using 10% fairness factor results in a remarkable reduction in standard deviation of completed tasks that implies increasing fairness. The standard deviation drops from 16% to 13.5%, at the cost of $\simeq$1% reduction in robustness (from 65% to 64%). This compromise in robustness is because deferring fewer tasks in an attempt to improve fairness results in fewer tasks successfully completed overall.

However, in the case of 30k oversubscription level (and to a lesser extent, 20k and 25k cases that are not shown in the figure), the fairness factor makes more significant differences as the oversubscription increases. This is due to the fact that a higher oversubscription level provides more tasks to select at each mapping event. Therefore there is more possibility to bias the mapping to make the task mapping fairer.

Since high fairness factor value significantly impact on robustness in highly oversubscribed cases, we configure PAMF with 10% fairness factor in the experiments, which include various oversubscription levels.

9.6. *Evaluating the Impact of Pruning Mechanism on The System Robustness*

In this experiment, we compare the overall robustness offered by PAM and PAMF against baseline heuristics described in Section 8 and those baseline heuristics retrofitted with probabilistic pruning mechanism. We conducted this evaluation under various oversubscription levels. However, for presentation clarity, we only show oversubscription levels with 15k and 30k tasks. We note that the same pattern is observed with other oversubscription levels evaluated.



(a) PAM and PAMF vs other heuristics. Horizontal axis shows oversubscription in form of number of tasks.

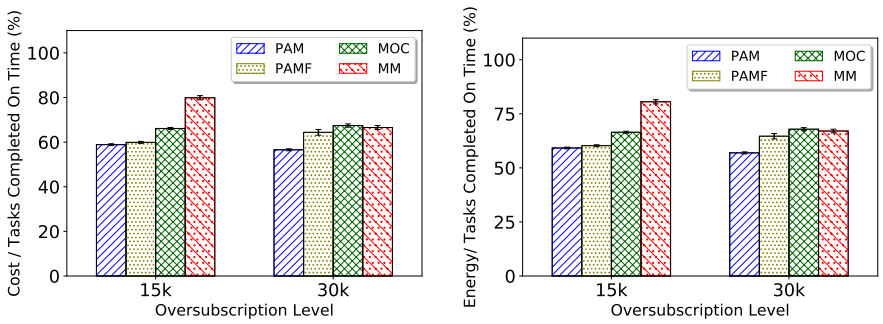
(b) Impact of pruning mechanism when attached to baseline mapping heuristics at 30k oversubscription level.

Figure 12: Comparison of PAM and PAMF against baseline heuristics with and without pruning mechanism. Vertical axis shows the percentage of task completed on time.

In Figure 12a, we observe that PAM results in a substantial increase in system robustness in comparison to other heuristics. On oversubscription level of 15k, PAM scores at nearly 67% robustness and PAMF, trading percentage of tasks completed for types of tasks completed, results in nearly 64% robustness. MOC, another heuristic that maps tasks based on the robustness value, is the closest in robustness to PAM, rivaling PAMF, at nearly 58%. The inability to probabilistically drop tasks leads to wasted processing and delayed task mapping, thereby lowering robustness. With robustness under 50%, the performance of MinMin lags behind, as it allocates tasks to machines no matter how unlikely they are to succeed. The robustness offered by both MSD and MMU suffers in comparison because these heuristics, instead of maximizing the performance of the most-likely tasks, prioritize tasks whose deadlines or urgency is closest (*i.e.,* least likely to succeed tasks). With an oversubscription of 30k tasks, MSD and MMU perform particularly bad because they mostly map tasks that fail to meet their deadlines. When comparing PAM and PAMF against the average of the other four heuristics, PAM and PAMF result in averagely 22% higher robustness.

In Figure 12b, we observe that, for all heuristics, adding the pruning mechanism to the existing mapping heuristics improves the robustness. The pruning mechanism makes the largest impact on MSD and MMU. These heuristics occasionally attempt to map tasks with too tight deadlines, thus, resulting in a low chance of success. By limiting these heuristics to map tasks whose chance is beyond a certain threshold, their overall system robustness is significantly improved.

9.7. Cost and Energy Gains of Probabilistic Task Pruning



(a) Analysis of the incurred cost

(b) Analysis of the consumed energy

Figure 13: Impact of probabilistic task pruning on the incurred cost and consumed energy of using resources. Horizontal axes show the oversubscription level and the vertical axes, respectively, show the average incurred cost and energy consumed per task completed on time.

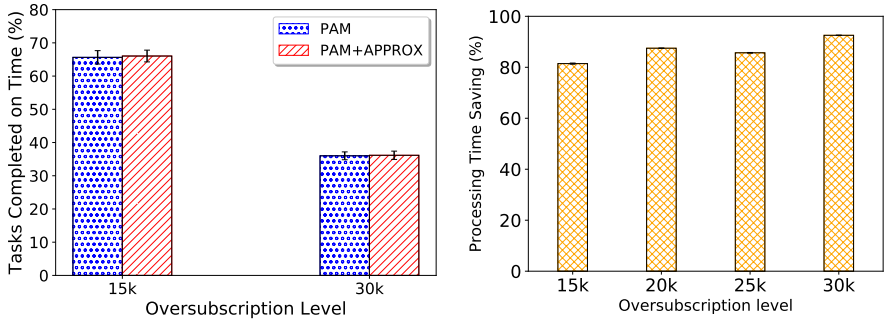
To investigate the incurred cost of using resources, pricing from Amazon cloud VMs [6] has been corresponded to the machines in the simulation. Energy consumption in active and idle states has been roughly estimated based on the machine's specification [41]. Specifically, we assume each machine to consume 70% of their rated power supply when the machine is active and 25% when it is idle. Each machine's usage time is tracked. The price and energy incurred to process the tasks are divided by the percentage of tasks completed on time to provide a normalized view of the incurred costs and consumed energy.

Figure 13a and 13b suggest that in an oversubscribed system, both PAM and PAMF incur at least 33% lower cost and energy per completed task than MM. We exclude MMU and MSD from the figure because they are shown to perform poorly in the prior experiment, which makes their cost per task completion ratio unchartable, when compared to other heuristics.

While previous tests have shown PAM outperforms other heuristics in terms of robustness in the face of oversubscription, these results demonstrate that the benefits are realized in dollar cost and consumed energy as well, due to not processing tasks needlessly.

9.8. Evaluating the Imposed Overhead

To evaluate task pruning mechanism and PAM's scheduling overhead. We compare PAM that is implemented from the concept introduced in Section 6 against PAM that utilizes computational-reuse and approximation techniques that we introduced in Section 7 (called approximate PAM and shown as *PAM+APPROX*). First, we compare the task mapping performance in terms of the number of tasks completed on time. Then, we measured and compared the makespan of the simulation, which is directly related to the scheduling overhead. For the sake of accuracy, all the measurements have been carried out on an isolated machine, without any disturbing workload.



(a) Analysis of obtained robustness

(b) Analysis of imposed overhead

Figure 14: Impact of reusing and approximation techniques of the pruning mechanism and mapping heuristics on the (a) system robustness and (b) imposed overhead. Horizontal axis in both figures shows the oversubscription level and vertical axis in (a) shows sytem robustness and in (b) shows the percentage of reduction in the imposed overhead.

Figure 14a shows that there is no statistically and practically significant difference in the performance of PAM and approximate PAM. This confirms our hypothesis that computational reuse does not change the mapping results, while the approximation introduces only minor rounding errors. However, due to the high uncertainty of the heterogeneous computing system, such approximated rounding induces only minimal changes to mapping decisions.

Figure 14b shows that the approximate PAM performs drastically faster than the PAM implementation. The saving is particularly remarkable on the larger oversubscription cases such as 30k where approximate PAM cuts the processing time out by 93% when compared to the PAM's implementation (*i.e.,* approximate PAM is 13.5 times faster than simple PAM). Another point we observe is the growth of execution time. PAM's scheduling overhead grows in a more than the linear way in response to the increase in oversubscription level. However, approximate PAM's scheduling overhead is a little lower on 30k than 15k oversubscription level. This is because while more oversubscribed workload puts more tasks in the batch queue at each moment (which make each mapping event slower), the higher oversubscription level also means there are fewer mapping events for the experiment with the same number of tasks arrival. The fewer mapping events and more load per mapping event cancel each other in the case of approximate PAM.

10. Conclusion and Future Works

The goal of this research was to improve robustness of HC systems via pruning tasks with low probability of success. We designed a pruning mechanism as part of resource allocation system. For pruning, we determined probability values to either defer or drop a task whose chance of success is low. We enabled the pruning mechanism to determine dropping threshold at the task level and dynamically adjust the deferring threshold based on the characteristics of the arriving workload. We developed a prob-

abilistic mapping heuristic, PAM, that cooperates with the pruning mechanism. We showed that PAM can improve system robustness by on average ≃22%. We upgraded PAM to accommodate fairness by compromising around four percentage points robustness. We employed approximate computing in calculation of probabilities in the system to reduce the scheduling and pruning overheads (by up to 93%) and ensure that the mechanism can be used practically. We concluded that: **(A)** when the system is not oversubscribed, tasks with low chance of success should be deferred (*i.e.,* wait for more favorable mapping in the next mapping); **(B)** When the system is sufficiently oversubscribed, the unlikely-to-succeed tasks must be dropped to alleviate the oversubscription and increase the probability of other tasks succeed; **(C)** The system benefits from setting higher deferring threshold than dropping threshold. Evaluation results revealed that the pruning mechanism (and PAM) not only improves system robustness but also reduces the cost and energy of using cloud-based HC systems by ≃33%.

The idea of pruning developed in this research is generic and can be plugged to other systems. We plan to extend the probabilistic approach for tasks preemption and its impact on the convolution process. Finally, as HC systems have various QoS concerns, domain-specific fairness models should be explored.

Acknowledgments

We thank reviewers of the journal. This is a substantially extended version of a paper presented at the 33rd IEEE International Parallel & Distributed Processing Symposium (IPDPS '19) [42]. Portions of this research were conducted with high performance computational resources provided by the Louisiana Optical Network Infrastructure [8]. This research was supported by the Louisiana Board of Regents under grant number LEQSF(2016-19)-RD-A-25.

References

[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. 3, no. 3, pp. 195–208, Nov. 2000.

[2] 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.

[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," *Jnl. of Systems Arch.*, 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] C.-H. Hong, I. Spence, and D. S. Nikolopoulos, "Gpu virtualization and scheduling methods: A comprehensive survey," *ACM Computing Surveys (CSUR)*, vol. 50, no. 3, pp. 35:1–35:37, Jun. 2017.

[6] "Amazon Web Sevices (AWS) Instance Types," https://aws.amazon.com/ec2/instance-types/, accessed May 03, 2018.

[7] 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.

[8] "Louisiana Optical Network Infrastructure (LONI) Resources QB2," http://hpc.loni.org/resources/hpc/system.php?system=QB2, accessed Aug 10, 2018.

[9] 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, Jul. 2009.

[10] Z. Wang, H. Qian, J. Li, and H. Chen, "Using restricted transactional memory to build a scalable in-memory database," in *Proc. of the 9th European Conference on Computer Systems*, 2014, pp. 26:1–26:15.

[11] J. C. Dos Anjos, M. D. de Assuncao, J. Bez, C. Geyer, E. P. De Freitas, A. Carissimi, J. P. C. Costa, G. Fedak, F. Freitag, V. Markl *et al.*, "Smart: An application framework for real time big data analysis on heterogeneous cloud environments," in *15th International Conference on Computer and Information Technology*, Oct. 2015, pp. 199–206.

[12] M. Malensek, S. Pallickara, and S. Pallickara, "Minerva: proactive disk scheduling for QoS in multitier, multitenant cloud environments," *IEEE Internet Computing*, vol. 20, no. 3, pp. 19–27, May. 2016.

[13] V. Shestak, J. Smith, A. Maciejewski, and H. J. Siegel, "Stochastic robustness metric and its use for static resource allocations," *Jnl. of Parallel and Distributed Computing*, vol. 68, no. 8, pp. 157–173, 2008.

[14] B. Khemka, R. Friese, L. D. Briceño, 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.

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

[16] S. AlEbrahim and I. Ahmad, "Task scheduling for heterogeneous computing systems," *Supercomputing Jnl.*, vol. 73, no. 6, Jun. 2017.

[17] R. Hussain, M. Amini, A. Kovalenko, Y. Feng, and O. Semiari, "Federated edge computing for disaster management in remote smart oil fields," in *Proceedings of the 21st IEEE International Conference on High Performance Computing and Communications*, ser. HPCC'19, Aug. 2019, pp. 929–936.

[18] C. Denninnart, J. Gentry, and M. A. Salehi, "Improving robustness of heterogeneous serverless computing systems via probabilistic task pruning," in *28th Heterogeneity in Computing Workshop (HCW 2019), in the proceedings of the IPDPS 2019 Workshops & PhD Forum (IPDPSW)*, May 2019.

[19] M. Salehi and R. Buyya, "Adapting market-oriented scheduling policies for cloud computing," in *Proceedings of the 10th international conference on Algorithms and Architectures for Parallel Processing-Volume Part I*, Jan. 2010, pp. 351–362.

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

[21] 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.

[22] 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.

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

[24] C. Delimitrou and C. Kozyrakis, "Quality-of-Service-aware scheduling in heterogeneous data centers with Paragon," *IEEE Micro*, vol. 34, no. 3, pp. 17–30, May 2014.

[25] X. Li, M. Amini Salehi, Y. Joshi, M. Darwich, L. Brad, and M. Bayoumi, "Performance analysis and modelling of video stream transcoding using heterogeneous cloud services," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 30, no. 4, pp. 910–922, Sep. 2018.

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

[27] 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 11th European Conference on Computer Systems*, Apr. 2016, pp. 35:1–35:16.

[28] 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, Oct. 2016, pp. 595–600.

[29] M. Darwich, M. A. Salehi, E. Beyazit, and M. Bayoumi, "Cost-efficient cloud-based video streaming through measuring hotness," *The Computer Journal*, vol. 62, no. 5, pp. 641–656, 2019.

[30] R. Aparicio-Pardo, K. Pires, A. Blanc, and G. Simon, "Transcoding live adaptive video streams at a massive scale in the cloud," in *Proceedings of the 6th ACM Multimedia Systems Conference*, Mar. 2015, pp. 49–60.

[31] X. Li, M. A. Salehi, and M. Bayoumi, "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, May 2016, pp. 600–603.

[32]

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

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

[35] 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.

[36] C. L. Bayes and M. D. Branco, "Bayesian inference for the skewness parameter of the scalar skew-normal distribution," *Brazilian Journal of Probability and Statistics*, pp. 141–163, Dec. 2007.

[37] W. M. Kader, H. Rashid, M. Mamun, and M. A. S. Bhuiyan, "Advancement of cmos schmitt trigger circuits," *Modern Applied Science*, vol. 6, no. 12, pp. 51–58, Nov. 2012.

[38] 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.

[39] M. Pedemonte, P. Ezzatti, and Á. Martín, "Accelerating the min-min heuristic," in *Parallel Processing and Applied Mathematics*, May 2016, vol. 11, pp. 101–110.

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

[41] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," *ACM SIGARCH computer architecture news*, vol. 35, no. 2, pp. 13–23, 2007.

[42] J. Gentry, C. Denninnart, and M. Amini Salehi, "Robust dynamic resource allocation via probabilistic task pruning in heterogeneous computing systems," in *Proceedings of the 33rd IEEE International Parallel & Distributed Processing Symposium*, ser. IPDPS '19, May 2019.