Robust Dynamic Resource Allocation via Probabilistic Task Pruning in Heterogeneous Computing Systems

James Gentry, Chavit Denninnart, Mohsen Amini Salehi
High Performance Cloud Computing (HPCC) Laboratory,
School of Computing and Informatics, University of Louisiana at Lafayette, USA
gentry@hpcclab.org, {cxd9974, amini}@louisiana.edu

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 and we find thresholds for dropping and deferring. Next, we develop a pruning-aware mapping heuristic and extend it to engender fairness across various task types. We show the cost benefit of using probabilistic pruning in an HC system. Simulation results, harnessing a selection of mapping heuristics, show efficacy of the pruning mechanism in improving robustness (on average by ≃25%) and cost in an oversubscribed HC system by up to ≃40%.

***Index Terms*—Heterogeneous Computing (HC), Probabilistic Pruning, Mapping Heuristic, Robustness.**

I. 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]–[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]–[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. 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 (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 behind 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.

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 of using resources. This is particularly important for users who deploy heterogeneous cloud VMs [17]. As such, we investigate the impact of the proposed probabilistic pruning mechanism on the incurred cost of using heterogeneous cloud VMs and compare it against common mapping methods.

In summary, the contributions of this paper are as follows:

- Mathematically modeling the impact of task dropping on the success probability of other tasks behind the dropped one.
- Developing a method to determine probablistic dropping and deferring thresholds in the pruning mechanism.
- Proposing a method to dynamically engage task dropping in response to oversubscription in an HC system.
- Developing a pruning-aware and a fairness-aware mapping heuristic for an HC system.
- Analyzing the cost benefit of the pruning mechanism.

Simulation results approve our hypotheses and show that the pruning mechanism can enhance robustness and the incurred cost. Importantly, the mechanism is more impactful under higher oversubscription levels. This paper is organized as follows. Section II situates this work in relation to existing literature. Section III establishes the problem and describes our system model. Then, Section V presents our proposed solution. Baseline heuristics are described in Section VI, along with the constraints and parameters of the experiment. Section VII presents and analyzes the simulation results. Finally, Section VIII concludes the paper and offers direction for future works.

II. Related Works

Mapping tasks in HC systems have been shown to be an NP-complete problem [18], [19]. As such, there are multiple prior efforts that achieve sub-optimal solutions and they are either influence or being similar to our work. Here are some notable mentions.

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 [20], 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 [21] 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 [22]. 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 system throughput.

In [23], 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.* [24] 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 [25] 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.

III. System Model

The motivation for this research comes from an HC system used for processing live video streaming services [26] (*e.g.,* YouTube Live and Twitch.tv [27]). In these services, video content is initially captured in a certain format and then processed (aka transcoded) to support diverse viewers' display devices [28]. As there is no value in executing live video streaming tasks that have missed their individual deadlines, they are dropped 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 [23]. 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.

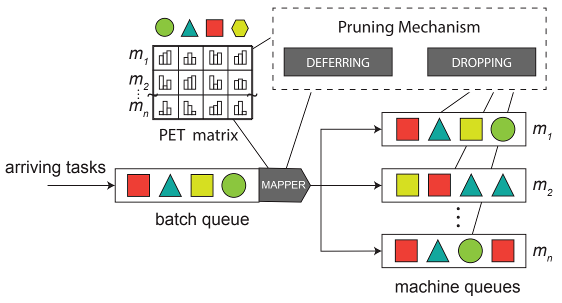


Fig. 1: Pruning mechanism. Heterogeneous tasks are mapped to heterogeneous machines in batches. In each mapping, the pruner drops or defers tasks based on their probability of success.

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], [20]. As we consider the HC systems are 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 [29]. 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. 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 [30], [31].

To map tasks to machines, the mapper creates a temporary (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.

IV. 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.,* robustness) 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 represents the completion time of task i on machine j at time t.

Let i 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 $PCT(i, j)$ [20]. Then, the robustness 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.

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

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 $PCT(i,j)$.

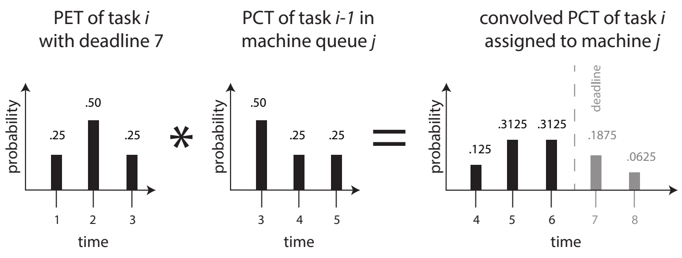


Fig. 2: Probabilistic Execution Time (PET) of arriving task i is convolved with the Probabilistic Completion Time (PCT) of the last task on machine j to form $PCT(i,j)$.

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 [20]. 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 [20] 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 $PCT(i-1,j)$.

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

(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 $PCT(i-1,j) \ge \delta_i$. 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 $t \ge \delta_i$, we use the helper Equation 3 to generate an impulse by discarding impulses of $PCT(i-1,j) \ge \delta_i$. 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.

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

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

(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 poses a not insignificant overhead. However, the overhead can be mitigated by pre-processing and memoizing portions of the convolution and finalizing it just in time at mapping time. It is also possible to approximate PMFs by aggregating impulses.

V. Maximizing Robustness via Pruning Mechanism

A. *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 (robustness) is calculated. Tasks whose robustness 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 set of questions surround the probability thresholds at which tasks are dropped or deferred. How to identify these thresholds and the relation between them. A related question 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. 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?

B. Determining Dropping and Deferring Thresholds

1) Dynamic Per-Task Dropping 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 can be taken into account to make the best decision about which tasks should stay and which are dropped.

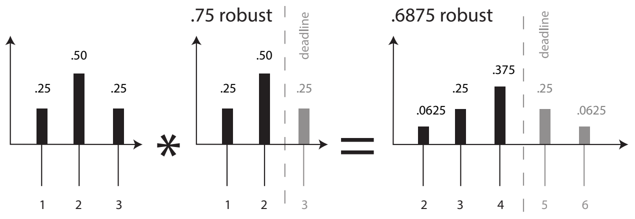
In addition to determining task robustness, other features of completion time PMF can be valuable in making decisions about probabilistic task dropping. We identify two task-level characteristics that further influence the robustness of tasks located behind a given task i: (A) the position of task i 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, 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.

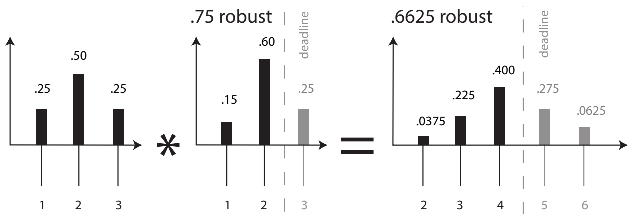
Skewness is defined as the measure of asymmetry in a probability distribution and is calculated based on Equation 6 [32]. In this equation, N is the sample size of a given PMF, Y_i is an observation, and $\bar{Y}$ is the mean of observations. A negative skewness value means the tail is on the left side 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 $-1 \leq s \leq 1$.

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

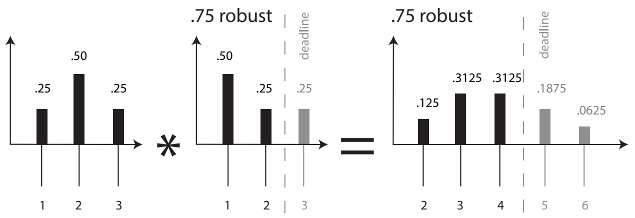
A negatively skewed PMF has the majority of probability occurring on the right side of PMF. Alternatively, because the bulk of a probability is biased to the left side of a PMF, a positive skew implies that a task is completed sooner than later. The middle PMFs in Figure 3 each represents a completion time with a robustness of 0.75, however, they show different types of skewness. Using this information, we can see that two tasks with the same robustness can have different impacts on the robustness 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 robustness of the next task in the queue.



(a) No Skew



(b) Left Skew



(c) Right Skew

Fig. 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 ($\delta_i = 3$).

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 threshold 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.

2) Inferring Deferring Threshold from Dropping Threshold: At its simplest, the pruner can use a single threshold to decide whether to defer mapping a task, as well as to decide whether to drop a mapped task. However, considering deferring threshold less than or equal to the dropping threshold, causes mapping a task with a robustness lower than the dropping threshold. Nonetheless, unless a task is dropped ahead of that mapped task, such a task is going to be dropped during the next engagement of the task dropper and leads to a poor performance. Therefore, considering deferring threshold *higher* than the dropping threshold benefits the overall robustness of the system. This is because the pruner waits for more robust mapping decisions and considers a gap between the requirements to map a task (*i.e.,* deferring threshold), and the threshold required to dropping a task. In Section VII-C, we explore the appropriate gap between deferring and dropping thresholds so that the robustness of HC system is maximized.

C. Aggressive Pruning by 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 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 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 8. In the experiment section, we analyze the impact of *lambda* and determine an appropriate value for it.

$$d_\tau = \mu_\tau \cdot \lambda + d_{\tau-1} \cdot (1-\lambda) \tag{8}$$

Another potential concern is minor fluctuations about the toggle switching the dropping off and then back on. We employ a Schmitt Trigger [33] 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.

D. Proposed Mapping Heuristics

In this part, we develop two mapping heuristics based on the theory explored in this study. The first heuristic, PAM, leverages pruning mechanism to maximize robustness. However, the second mapping heuristic, in addition to maximizing robustness, aims at achieving fairness across task types.

The batch heuristics are two-phase processes, 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 chooses the best task-machine pair for each available machine queue slot. After all slots are filled, virtual mappings are assigned to the machine queues and the mapping method is complete.

1) Pruning Aware Mapper (PAM): This heuristic uses the PET matrix to calculate task robustness and then operates based on the pruning mechanism. Before making any mapping decision, PAM analyzes the oversubscription level and performs task dropping on machine queues, if necessary.

In the first phase, for each unmapped task, PAM finds the machine offers the highest robustness. Then, tasks that do not meet the deferring threshold are pruned. The second phase finds the task-machine pair with the lowest completion time and maps it to that machine's virtual queue. Ties are broken by choosing task with the shortest expected execution time.

2) Fair Pruning 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 heuristic aims at mitigating this unfairness.

PAMF favors task types that have suffered from pruning. By relaxing the pruning thresholds for tasks of unfairly treated task types, the system can prevent bias against them. We define *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.,* $\varepsilon_{ef} = \varepsilon_{(e-1)f} + \vartheta$. We note that we limit sufferage values (ε_{ef}) to be between 0 to 100%. Then, 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.

Updated pruning threshold enables PAMF create 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.

VI. Experimental Setup

A. 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 machines[1] was 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.

B. 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 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 other workload investigations [14], [15], 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. Except an experiment in Subsection VII-G, 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. 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.

[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.

C. Baseline Mapping Heuristics

1) MinCompletion-MinCompletion (MM): This heuristic has been extensively used in the literature [20], [34]–[36]. 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 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.

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.

3) MinCompletion-MaxUrgency (MMU): Urgency of task i on machine j is defined as $U_{ij} = 1/(\delta_i - \mathbf{E}(C_{ij}))$, where $\mathbf{E}(C_{ij})$ 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.

4) Max Ontime Completions (MOC): The MOC heuristic was developed in [20]. 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.

VII. Performance Evaluation

A. 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 800 tasks per trial. Each of the experiments investigates extreme levels of oversubscription where few tasks complete successfully using baseline heuristics. Each machine in the HC system has a machine-queue size of six, counting the executing task and the dropping toggle is one task. 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).

B. Dynamic engagement of probabilistic task dropping

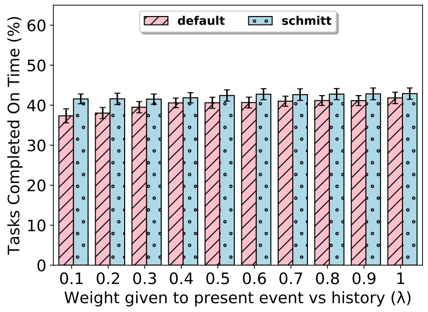


Fig. 4: Impact of historical oversubscription observations and Schmitt Trigger on determining oversubscription level of HC system.

In this experiment, our aim is to appropriately configure oversubscription level (see Equation 8) 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 34k tasks arriving to the system.

Figure 4 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 67.7% to 71.9%. This is due in part to the steady nature of task-arrival our workload trials with only few sudden spikes. Also, we can observe that applying Schmitt Trigger results in a higher robustness. Specifically, we observe that $\lambda = 0.9$ provides a statistically and practically higher robustness, hence, is appropriate for identifying oversubscription level.

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).

C. Impact of deferring and dropping thresholds

The goal of this experiment is to identify the appropriate gap that should be considered between the deferring and dropping thresholds (see Section V-B). To find the appropriate deferring threshold, we add a gap value to the dropping threshold (*e.g.,* a dropping threshold of 50% would require at least 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 gap on each by 5% until the deferring threshold reaches 90%. The results, shown in Figure 5, are generated from an a workload with 34k tasks.

Figure 5 validates our hypothesis by showing that using a higher deferring threshold leads to higher system robustness. We observe that if the deferring threshold is chosen high enough, the impact of dropping threshold on the system robustness diminishes. That is, with a high dropping threshold

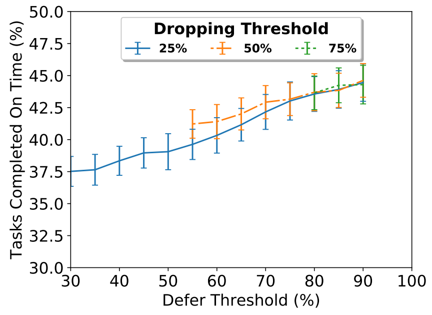


Fig. 5: Impact of deferring and dropping thresholds. Dropping threshold is denoted by line type and color.

(*e.g.,* 75%) we obtain the same system robustness with a low dropping threshold (*e.g.,* 25%). However, we note that higher dropping threshold can influence 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. In the rest of experiments, dropping threshold 50% and deferring threshold 90% is used.

D. Evaluating the impact of fairness factor

Our aim is to study if PAMF heuristic (see Section V) alleviates unfairness. We test the system using a fairness factor ranging from 0% (*i.e.,* no fairness) 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 6. We tested oversubscription level of 19k and 34k tasks.

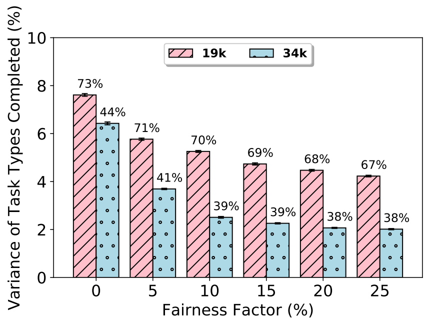


Fig. 6: Evaluating fairness and robustness. Horizontal axis shows fairness factor modifier to the sufferage value. Vertical axis is the variance of completed task types. Values above bars show robustness.

Figure 6 shows that significant improvement in fairness can be attained at the cost of compromising robustness. In particular, we observe that using 5% fairness factor results in remarkable reduction in variance of completed tasks that implies increasing fairness. For instance, for 34k tasks, the variance drops from 6% to 3.5%, at a cost of $\simeq$10% reduction in robustness (from 44.2% to 40.5%). This compromise in robustness is because deferring fewer tasks in an attempt to improve fairness results in fewer tasks successfully completed overall. Further increasing fairness factor results in insignificant changes on fairness and robustness, therefore, we configure PAMF with 5% fairness factor in the experiments.

E. Evaluating robustness of pruning

In this experiment, we compare the overall robustness offered by PAM and PAMF against that of the baseline heuristics described in section VI. We conducted this evaluation under various oversubscription levels, however, due to space limit and presentation clarity we only show oversubscription levels with 19k and 34k tasks. We note that the same pattern is observed with other oversubscription levels evaluated.

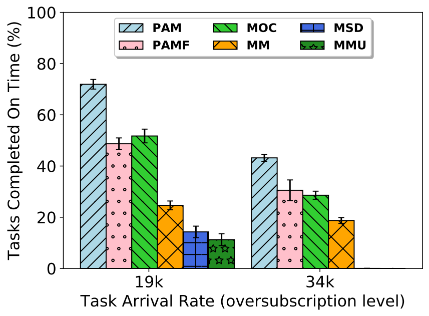


Fig. 7: Comparison of PAM and PAMF against other heuristics. Horizontal axis shows oversubscription in form of number of tasks.

In Figure 7, we observe that PAM results in a substantial increase in system robustness versus other heuristics, at nearly 70% and PAMF results in nearly 50% robustness, trading percentage of tasks completed for types of tasks completed. MOC, another robustness-based heuristic, is the closest in robustness to PAM, rivaling PAMF, at nearly 50%. The inability to probabilistically drop tasks leads to wasted processing and delayed task mapping, thereby lowering robustness. With robustness of $\simeq$25%, the performance of MinMin lags far 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 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 34k tasks, MSD and MMU only map tasks that fail to meet their deadline.

F. Cost benefit of probabilistic pruning

To investigate the incurred cost of using resources, pricing from Amazon cloud VMs [6] 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 provide a normalized view of the incurred costs in the system.

Figure 8 shows that in an oversubscribed system both PAM and PAMF incur a significantly ($\simeq$40%) lower cost per completed task than MOC and other heuristics. At extreme levels of oversubscription, the difference between heuristics such as MMU and MSD and PAM become unchartable, as MSD and

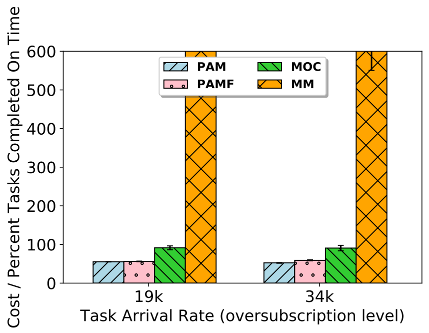


Fig. 8: Impact of probabilistic pruning on incurred costs of using resources. Horizontal axis shows the oversubscription level.

MMU both prioritize tasks least likely to succeed, whereas PAM prioritizes those most likely to succeed. While previous tests have shown PAM outperforms 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.

G. Evaluating video transcoding workload traces

To evaluate PAMF under a real-world setting, we compare it against MinMin on video transcoding workload traces under different oversubscription levels (horizontal axis in Figure 9). The PET matrix captured from running four video transcoding types on 660 video files (available in https://goo.gl/TE5iJ5) on four heterogeneous Amazon EC2 VMs, namely CPU-Optimized, Memory-Optimized, General Purpose, and GPU. The experimental result confirms our earlier observation and show that PAMF outperforms MinMin specifically as the level of oversubscription increases.

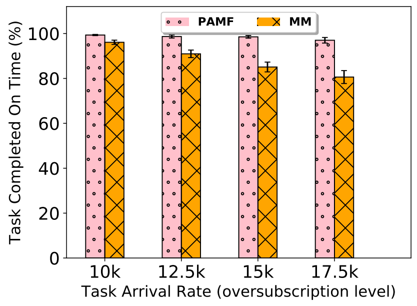


Fig. 9: Comparison of PAMF against MinMin using video transcoding workload. The horizontal axis shows the oversubscription level (number of tasks).

VIII. 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 in the system. For pruning, we determined probability values used by mapping heuristics to either map or defer a task. 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. We developed a mapping heuristic, PAM, based on the probabilistic task pruning and showed that it can improve system robustness by on average by $\simeq$25%. We upgraded PAM to accommodate fairness by compromising around four percentage points robustness. Evaluation results revealed that pruning mechanism (and PAM) does not only improve system robustness but also reduces the cost of using cloud-based HC systems by $\simeq$40%.

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. Another future work is to approximately compute tasks, in addition to pruning or dropping them from the system. Finally, as HC systems have various QoS concerns, domain-specific fairness models should be explored.

ACKNOWLEDGMENTS

We thank reviewers of the paper. 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. 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.

[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] 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.

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

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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," *Accepted in IEEE Transactions on Parallel and Distributed Systems (TPDS)*, Sep. 2018.

[24] 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.

[25] 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.

[26] 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. BD-Cloud '16, Oct. 2016, pp. 595–600.

[27] 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.

[28] 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.

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

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.